

Prototyping Board Manual

**AMI[®] 6800
Microprocessors**

Table of Contents

INTRODUCTION	III	CHAPTER 4	
CHAPTER 1		REENTRANT SELF-RELATIVE SUBROUTINE	
SYSTEM DESCRIPTION	1	ROMs (RSRSRS) = (RS) ³	13
MPU	1	CONCEPTS	13
Clock	2	IMPLEMENTATION	13
Reset	2	THE PROTO SYSTEM	14
Restart	2	(RS) ³ IN PROTO — SUBROUTINE	
BUS STRUCTURE	2	DESCRIPTIONS	14
MEMORY ADDRESS ASSIGNMENT ..	3	CHAPTER 5	
READ ONLY MEMORY	3	OPERATING PROCEDURES	19
RANDOM ACCESS MEMORY	3	CHAPTER 6	
PARALLEL I/O	3	TROUBLESHOOTING PROCEDURES	21
SERIAL I/O	3	APPENDIX A	
EPROM PROGRAMMER	4	6800 HEX TAPE FORMAT	A-1
INTERVAL TIMER	4	APPENDIX B	
DMA	4	PARTS LIST FOR 6800 PROTOTYPING	
CHAPTER 2		BOARD	B-1
PHYSICAL DESCRIPTION	5	APPENDIX C	
CHAPTER 3		PROGRAM LISTING	C-1
SOFTWARE	9	PROTO (PROTOTYPE BOARD	
L, ADDL, ADDH, OFFSET	9	MONITOR PROGRAM)	C-1
P, ADDL, ADDH, OFFSET	9	PROM BURNER ADDITION TO	
S, ADD BYTE 1, BYTE 2, ---, BYTEN .	10	PROTO	C-22
D, ADDL, ADDH	10	RSRSRS—REENTRANT SELF-RELA-	
G, ADDR	10	TIVE SUBROUTINE ROMs	C-29
R	10	PROTOTYPE SYSTEM SOFTWARE IN	
B, ADDL, ADDH, ROMAD	10	HEX TAPE FORMAT	C-48
V, ADDL, ADDH, ROMAD	11	APPENDIX D	
I, ADDL, ADDH, ROMAD	11	S6800 INSTRUCTION SET	D-1
M, ADDL, ADDH, DEST	11	APPENDIX E	
THE SUBROUTINE ROM	11	AMI SALES OFFICES	E-1
INTERRUPTS	11	APPENDIX F	
BREAKPOINTS	12	PCB COMPONENT LAYOUT, ARTWORK	
		AND SCHEMATIC DIAGRAMS	F-1

List of Figures & Tables

Figure 1-1.	AMI 6800 Prototyping Board Block Diagram.	1
Figure 1-2.	Memory Assignment Map for the AMI 6800 Prototyping Board	2
Figure 5-1.	AMI 6800 Prototyping Board Switch Configuration	20
Table 1-1.	I/O Address Assignment	3
Table 1-2.	Bit Rate Generator Switch Settings	4
Table 2-1.	I/O Pin Assignments	5
Table 5-1.	Prototyping Board Current Requirements	19

Introduction

The AMI6800 Prototyping Board is a single board, hardware and software, prototyping system. It allows system development using a functionally compatible system for reduced development time. With this board the basic 6800 family parts (S6800, S6810, S6820, S6830, S6831, S6834 and S6850) can be evaluated. It may also serve as a general purpose microcomputer for low volume systems to which the user can easily add I/O or memory. The 10½" x 12" card has two 86 pin edge connectors, one for MPU Bus lines and one for I/O.

The AMI 6800 Prototyping Board has several major features which offer great flexibility for program development.

MAJOR FEATURES

- 2K Bytes ROM
- 512 Bytes EPROM
- Location for 2K Bytes EPROM
- 1K Bytes RAM

- EPROM Programming Capability for S6834
 - Variable Speed or Crystal Controlled Clock
 - Up to 58 Programmable I/O lines
 - ACIA with either TTY or RS-232C Interface
 - Single +5 Volt Power Source except when using EPROM or RS-232C Interface Chips
 - Totally Buffered MPU Lines Available at Edge Connector Pins
 - RAM in High Order Memory for Dynamic Interrupt Flexibility
 - Interval Timer Giving 1ms and 100μs Interrupts
 - Selectable DMA Mode
 - Restart Address Selection
 - TTY Operating System Software with a Re-Entrant Self-Relative Subroutine
 - ROM and an EPROM Programming Subroutine
- Several of the above options are switch selectable. See Chapter 5 for details.

Introduction

- EPROM Programming Capability for 28834
 - Variable Speed or Crystal Controlled Clock
 - Up to 64 Programmable I/O Pins
 - ACTA with either TTY or RS-232C Interface
 - Single +5 Volt Power Source except when using EPROM or RS-232C Interface Cards
 - Totally Buffered MPU Lines Available at Edge Connector Pins
 - RAM in High Order Memory for Expansion
 - Interrupt Flexibility
 - Interval Timer Giving 1ms and 100µs Intervals
 - Selectable DMA Mode
 - Restart Address Selection
 - TTY Operating System Software with a Permanent Self-Reliance Subroutine
 - ROM and an EPROM Programming Subroutine
- Several of the above options are switch selectable. See Chapter 5 for details.

The AMI 6800 Prototyping Board is a single board hardware and software prototyping system. It allows system development using a functionally compatible system for reduced development time. With this board the basic 6800 family parts (6800, 6801, 6802, 6803, 6804, 6805, 6806, 6807, 6808, 6809, 6810, 6811, 6812, 6813, 6814, 6815, 6816, 6817, 6818, 6819, 6820, 6821, 6822, 6823, 6824, 6825, 6826, 6827, 6828, 6829, 6830, 6831, 6832, 6833, 6834, 6835, 6836, 6837, 6838, 6839, 6840, 6841, 6842, 6843, 6844, 6845, 6846, 6847, 6848, 6849, 6850, 6851, 6852, 6853, 6854, 6855, 6856, 6857, 6858, 6859, 6860, 6861, 6862, 6863, 6864, 6865, 6866, 6867, 6868, 6869, 6870, 6871, 6872, 6873, 6874, 6875, 6876, 6877, 6878, 6879, 6880, 6881, 6882, 6883, 6884, 6885, 6886, 6887, 6888, 6889, 6890, 6891, 6892, 6893, 6894, 6895, 6896, 6897, 6898, 6899, 6900, 6901, 6902, 6903, 6904, 6905, 6906, 6907, 6908, 6909, 6910, 6911, 6912, 6913, 6914, 6915, 6916, 6917, 6918, 6919, 6920, 6921, 6922, 6923, 6924, 6925, 6926, 6927, 6928, 6929, 6930, 6931, 6932, 6933, 6934, 6935, 6936, 6937, 6938, 6939, 6940, 6941, 6942, 6943, 6944, 6945, 6946, 6947, 6948, 6949, 6950, 6951, 6952, 6953, 6954, 6955, 6956, 6957, 6958, 6959, 6960, 6961, 6962, 6963, 6964, 6965, 6966, 6967, 6968, 6969, 6970, 6971, 6972, 6973, 6974, 6975, 6976, 6977, 6978, 6979, 6980, 6981, 6982, 6983, 6984, 6985, 6986, 6987, 6988, 6989, 6990, 6991, 6992, 6993, 6994, 6995, 6996, 6997, 6998, 6999, 7000, 7001, 7002, 7003, 7004, 7005, 7006, 7007, 7008, 7009, 7010, 7011, 7012, 7013, 7014, 7015, 7016, 7017, 7018, 7019, 7020, 7021, 7022, 7023, 7024, 7025, 7026, 7027, 7028, 7029, 7030, 7031, 7032, 7033, 7034, 7035, 7036, 7037, 7038, 7039, 7040, 7041, 7042, 7043, 7044, 7045, 7046, 7047, 7048, 7049, 7050, 7051, 7052, 7053, 7054, 7055, 7056, 7057, 7058, 7059, 7060, 7061, 7062, 7063, 7064, 7065, 7066, 7067, 7068, 7069, 7070, 7071, 7072, 7073, 7074, 7075, 7076, 7077, 7078, 7079, 7080, 7081, 7082, 7083, 7084, 7085, 7086, 7087, 7088, 7089, 7090, 7091, 7092, 7093, 7094, 7095, 7096, 7097, 7098, 7099, 7100, 7101, 7102, 7103, 7104, 7105, 7106, 7107, 7108, 7109, 7110, 7111, 7112, 7113, 7114, 7115, 7116, 7117, 7118, 7119, 7120, 7121, 7122, 7123, 7124, 7125, 7126, 7127, 7128, 7129, 7130, 7131, 7132, 7133, 7134, 7135, 7136, 7137, 7138, 7139, 7140, 7141, 7142, 7143, 7144, 7145, 7146, 7147, 7148, 7149, 7150, 7151, 7152, 7153, 7154, 7155, 7156, 7157, 7158, 7159, 7160, 7161, 7162, 7163, 7164, 7165, 7166, 7167, 7168, 7169, 7170, 7171, 7172, 7173, 7174, 7175, 7176, 7177, 7178, 7179, 7180, 7181, 7182, 7183, 7184, 7185, 7186, 7187, 7188, 7189, 7190, 7191, 7192, 7193, 7194, 7195, 7196, 7197, 7198, 7199, 7200, 7201, 7202, 7203, 7204, 7205, 7206, 7207, 7208, 7209, 7210, 7211, 7212, 7213, 7214, 7215, 7216, 7217, 7218, 7219, 7220, 7221, 7222, 7223, 7224, 7225, 7226, 7227, 7228, 7229, 7230, 7231, 7232, 7233, 7234, 7235, 7236, 7237, 7238, 7239, 7240, 7241, 7242, 7243, 7244, 7245, 7246, 7247, 7248, 7249, 7250, 7251, 7252, 7253, 7254, 7255, 7256, 7257, 7258, 7259, 7260, 7261, 7262, 7263, 7264, 7265, 7266, 7267, 7268, 7269, 7270, 7271, 7272, 7273, 7274, 7275, 7276, 7277, 7278, 7279, 7280, 7281, 7282, 7283, 7284, 7285, 7286, 7287, 7288, 7289, 7290, 7291, 7292, 7293, 7294, 7295, 7296, 7297, 7298, 7299, 7300, 7301, 7302, 7303, 7304, 7305, 7306, 7307, 7308, 7309, 7310, 7311, 7312, 7313, 7314, 7315, 7316, 7317, 7318, 7319, 7320, 7321, 7322, 7323, 7324, 7325, 7326, 7327, 7328, 7329, 7330, 7331, 7332, 7333, 7334, 7335, 7336, 7337, 7338, 7339, 7340, 7341, 7342, 7343, 7344, 7345, 7346, 7347, 7348, 7349, 7350, 7351, 7352, 7353, 7354, 7355, 7356, 7357, 7358, 7359, 7360, 7361, 7362, 7363, 7364, 7365, 7366, 7367, 7368, 7369, 7370, 7371, 7372, 7373, 7374, 7375, 7376, 7377, 7378, 7379, 7380, 7381, 7382, 7383, 7384, 7385, 7386, 7387, 7388, 7389, 7390, 7391, 7392, 7393, 7394, 7395, 7396, 7397, 7398, 7399, 7400, 7401, 7402, 7403, 7404, 7405, 7406, 7407, 7408, 7409, 7410, 7411, 7412, 7413, 7414, 7415, 7416, 7417, 7418, 7419, 7420, 7421, 7422, 7423, 7424, 7425, 7426, 7427, 7428, 7429, 7430, 7431, 7432, 7433, 7434, 7435, 7436, 7437, 7438, 7439, 7440, 7441, 7442, 7443, 7444, 7445, 7446, 7447, 7448, 7449, 7450, 7451, 7452, 7453, 7454, 7455, 7456, 7457, 7458, 7459, 7460, 7461, 7462, 7463, 7464, 7465, 7466, 7467, 7468, 7469, 7470, 7471, 7472, 7473, 7474, 7475, 7476, 7477, 7478, 7479, 7480, 7481, 7482, 7483, 7484, 7485, 7486, 7487, 7488, 7489, 7490, 7491, 7492, 7493, 7494, 7495, 7496, 7497, 7498, 7499, 7500, 7501, 7502, 7503, 7504, 7505, 7506, 7507, 7508, 7509, 7510, 7511, 7512, 7513, 7514, 7515, 7516, 7517, 7518, 7519, 7520, 7521, 7522, 7523, 7524, 7525, 7526, 7527, 7528, 7529, 7530, 7531, 7532, 7533, 7534, 7535, 7536, 7537, 7538, 7539, 7540, 7541, 7542, 7543, 7544, 7545, 7546, 7547, 7548, 7549, 7550, 7551, 7552, 7553, 7554, 7555, 7556, 7557, 7558, 7559, 7560, 7561, 7562, 7563, 7564, 7565, 7566, 7567, 7568, 7569, 7570, 7571, 7572, 7573, 7574, 7575, 7576, 7577, 7578, 7579, 7580, 7581, 7582, 7583, 7584, 7585, 7586, 7587, 7588, 7589, 7590, 7591, 7592, 7593, 7594, 7595, 7596, 7597, 7598, 7599, 7600, 7601, 7602, 7603, 7604, 7605, 7606, 7607, 7608, 7609, 7610, 7611, 7612, 7613, 7614, 7615, 7616, 7617, 7618, 7619, 7620, 7621, 7622, 7623, 7624, 7625, 7626, 7627, 7628, 7629, 7630, 7631, 7632, 7633, 7634, 7635, 7636, 7637, 7638, 7639, 7640, 7641, 7642, 7643, 7644, 7645, 7646, 7647, 7648, 7649, 7650, 7651, 7652, 7653, 7654, 7655, 7656, 7657, 7658, 7659, 7660, 7661, 7662, 7663, 7664, 7665, 7666, 7667, 7668, 7669, 7670, 7671, 7672, 7673, 7674, 7675, 7676, 7677, 7678, 7679, 7680, 7681, 7682, 7683, 7684, 7685, 7686, 7687, 7688, 7689, 7690, 7691, 7692, 7693, 7694, 7695, 7696, 7697, 7698, 7699, 7700, 7701, 7702, 7703, 7704, 7705, 7706, 7707, 7708, 7709, 7710, 7711, 7712, 7713, 7714, 7715, 7716, 7717, 7718, 7719, 7720, 7721, 7722, 7723, 7724, 7725, 7726, 7727, 7728, 7729, 7730, 7731, 7732, 7733, 7734, 7735, 7736, 7737, 7738, 7739, 7740, 7741, 7742, 7743, 7744, 7745, 7746, 7747, 7748, 7749, 7750, 7751, 7752, 7753, 7754, 7755, 7756, 7757, 7758, 7759, 7760, 7761, 7762, 7763, 7764, 7765, 7766, 7767, 7768, 7769, 7770, 7771, 7772, 7773, 7774, 7775, 7776, 7777, 7778, 7779, 7780, 7781, 7782, 7783, 7784, 7785, 7786, 7787, 7788, 7789, 7790, 7791, 7792, 7793, 7794, 7795, 7796, 7797, 7798, 7799, 7800, 7801, 7802, 7803, 7804, 7805, 7806, 7807, 7808, 7809, 7810, 7811, 7812, 7813, 7814, 7815, 7816, 7817, 7818, 7819, 7820, 7821, 7822, 7823, 7824, 7825, 7826, 7827, 7828, 7829, 7830, 7831, 7832, 7833, 7834, 7835, 7836, 7837, 7838, 7839, 7840, 7841, 7842, 7843, 7844, 7845, 7846, 7847, 7848, 7849, 7850, 7851, 7852, 7853, 7854, 7855, 7856, 7857, 7858, 7859, 7860, 7861, 7862, 7863, 7864, 7865, 7866, 7867, 7868, 7869, 7870, 7871, 7872, 7873, 7874, 7875, 7876, 7877, 7878, 7879, 7880, 7881, 7882, 7883, 7884, 7885, 7886, 7887, 7888, 7889, 7890, 7891, 7892, 7893, 7894, 7895, 7896, 7897, 7898, 7899, 7900, 7901, 7902, 7903, 7904, 7905, 7906, 7907, 7908, 7909, 7910, 7911, 7912, 7913, 7914, 7915, 7916, 7917, 7918, 7919, 7920, 7921, 7922, 7923, 7924, 7925, 7926, 7927, 7928, 7929, 7930, 7931, 7932, 7933, 7934, 7935, 7936, 7937, 7938, 7939, 7940, 7941, 7942, 7943, 7944, 7945, 7946, 7947, 7948, 7949, 7950, 7951, 7952, 7953, 7954, 7955, 7956, 7957, 7958, 7959, 7960, 7961, 7962, 7963, 7964, 7965, 7966, 7967, 7968, 7969, 7970, 7971, 7972, 7973, 7974, 7975, 7976, 7977, 7978, 7979, 7980, 7981, 7982, 7983, 7984, 7985, 7986, 7987, 7988, 7989, 7990, 7991, 7992, 7993, 7994, 7995, 7996, 7997, 7998, 7999, 8000, 8001, 8002, 8003, 8004, 8005, 8006, 8007, 8008, 8009, 8010, 8011, 8012, 8013, 8014, 8015, 8016, 8017, 8018, 8019, 8020, 8021, 8022, 8023, 8024, 8025, 8026, 8027, 8028, 8029, 8030, 8031, 8032, 8033, 8034, 8035, 8036, 8037, 8038, 8039, 8040, 8041, 8042, 8043, 8044, 8045, 8046, 8047, 8048, 8049, 8050, 8051, 8052, 8053, 8054, 8055, 8056, 8057, 8058, 8059, 8060, 8061, 8062, 8063, 8064, 8065, 8066, 8067, 8068, 8069, 8070, 8071, 8072, 8073, 8074, 8075, 8076, 8077, 8078, 8079, 8080, 8081, 8082, 8083, 8084, 8085, 8086, 8087, 8088, 8089, 8090, 8091, 8092, 8093, 8094, 8095, 8096, 8097, 8098, 8099, 8100, 8101, 8102, 8103, 8104, 8105, 8106, 8107, 8108, 8109, 8110, 8111, 8112, 8113, 8114, 8115, 8116, 8117, 8118, 8119, 8120, 8121, 8122, 8123, 8124, 8125, 8126, 8127, 8128, 8129, 8130, 8131, 8132, 8133, 8134, 8135, 8136, 8137, 8138, 8139, 8140, 8141, 8142, 8143, 8144, 8145, 8146, 8147, 8148, 8149, 8150, 8151, 8152, 8153, 8154, 8155, 8156, 8157, 8158, 8159, 8160, 8161, 8162, 8163, 8164, 8165, 8166, 8167, 8168, 8169, 8170, 8171, 8172, 8173, 8174, 8175, 8176, 8177, 8178, 8179, 8180, 8181, 8182, 8183, 8184, 8185, 8186, 8187, 8188, 8189, 8190, 8191, 8192, 8193, 8194, 8195, 8196, 8197, 8198, 8199, 8200, 8201, 8202, 8203, 8204, 8205, 8206, 8207, 8208, 8209, 8210, 8211, 8212, 8213, 8214, 8215, 8216, 8217, 8218, 8219, 8220, 8221, 8222, 8223, 8224, 8225, 8226, 8227, 8228, 8229, 8230, 8231, 8232, 8233, 8234, 8235, 8236, 8237, 8238, 8239, 8240, 8241, 8242, 8243, 8244, 8245, 8246, 8247, 8248, 8249, 8250, 8251, 8252, 8253, 8254, 8255, 8256, 8257, 8258, 8259, 8260, 8261, 8262, 8263, 8264, 8265, 8266, 8267, 8268, 8269, 8270, 8271, 8272, 8273, 8274, 8275, 8276, 8277, 8278, 8279, 8280, 8281, 8282, 8283, 8284, 8285, 8286, 8287, 8288, 8289, 8290, 8291, 8292, 8293, 8294, 8295, 8296, 8297, 8298, 8299, 8300, 8301, 8302, 8303, 8304, 8305, 8306, 8307, 8308, 8309, 8310, 8311, 8312, 8313, 8314, 8315, 8316, 8317, 8318, 8319, 8320, 8321, 8322, 8323, 8324, 8325, 8326, 8327, 8328, 8329, 8330, 8331, 8332, 8333, 8334, 8335, 8336, 8337, 8338, 8339, 8340, 8341, 8342, 8343, 8344, 8345, 8346, 8347, 8348, 8349, 8350, 8351, 8352, 8353, 8354, 8355, 8356, 8357, 8358, 8359, 8360, 8361, 8362, 8363, 8364, 8365, 8366, 8367, 8368, 8369, 8370, 8371, 8372, 8373, 8374, 8375, 8376, 8377, 8378, 8379, 8380, 8381, 8382, 8383, 8384, 8385, 8386, 8387, 8388, 8389, 8390, 8391, 8392, 8393, 8394, 8395, 8396, 8397, 8398, 8399, 8400, 8401, 8402, 8403, 8404, 8405, 8406, 8407, 8408, 8409, 8410, 8411, 8412, 8413, 8414, 8415, 8416, 8417, 8418, 8419, 8420, 8421, 8422, 8423, 8424, 8425, 8426, 8427, 8428, 8429, 8430, 8431, 8432, 8433, 8434, 8435, 8436, 8437, 8438, 8439, 8440, 8441, 8442, 8443, 8444, 8445, 8446, 8447, 8448, 8449, 8450, 8451, 8452, 8453, 8454, 8455, 8456, 8457, 8458, 8459, 8460, 8461, 8462, 8463, 8464, 8465, 8466, 8467, 8468, 8469, 8470, 8471, 8472, 8473, 8474, 8475, 8476, 8477, 8478, 8479, 8480, 8481, 8482, 8483, 8484, 8485, 8486, 8487, 8488, 8489, 8490, 8491, 8492, 8493, 8494, 8495, 8496, 8497, 8498, 8499, 8500, 8501, 8502, 8503, 8504, 8505, 8506, 8507, 8508, 8509, 8510, 8511, 8512, 8513, 8514, 8515, 8516, 8517, 8518, 8519, 8520, 8521, 8522, 8523, 8524, 8525, 8526, 8527, 8528, 8529, 8530, 8531, 8532, 8533, 8534, 8535, 8536, 8537, 8538, 8539, 8540, 8541, 8542, 8543, 8544, 8545, 8546, 8547, 8548, 8549, 8550, 8551, 8552, 8553, 8554, 8555, 8556, 8557, 8558, 8559, 8560, 8561, 8562, 8563, 8564, 8565, 8566, 8567, 8568, 8569, 8570, 8571, 8572, 8573, 8574, 8575, 8576, 8577, 8578, 8579, 8580, 8581, 8582, 8583, 8584, 8585, 8586, 8587, 8588, 8589, 8590, 8591, 8592, 8593, 8594, 8595, 8596, 8597, 8598, 8599, 8600, 8601, 8602, 8603, 8604, 8605, 8606, 8607, 8608, 8609, 8610, 8611, 8612, 8613, 8614, 8615, 8616, 8617, 8618, 8619, 8620, 8621, 8622, 8623, 8624, 8625, 8626, 8627, 8628, 8629, 8630, 8631, 8632, 8633, 8634, 8635, 8636, 8637, 8638, 8639, 8640, 8641, 8642, 8643, 8644, 8645, 8646, 8647, 8648, 8649, 8650, 8651, 8652, 8653, 8654, 8655, 8656, 8657, 8658, 8659, 8660, 8661, 8662, 8663, 8664, 8665, 8666, 8667, 8668, 8669, 8670, 8671, 8672, 8673, 8674, 8675, 8676, 8677, 8678, 8679, 8680, 8681, 8682, 8683, 8684, 8685, 8686, 8687, 8688, 8689, 8690, 8691, 8692, 8693, 8694, 8695, 8696, 8697, 8698, 8699, 8700, 8701, 8702, 8703, 8704, 8705, 8706, 8707, 8708, 8709, 8710, 8711, 8712, 8713, 8714, 8715, 8716, 8717, 8718, 8719, 8720, 8721, 8722, 8723, 8724, 8725, 8726, 8727, 8728, 8729, 8730, 8731, 8732, 8733, 8734, 8735, 8736, 8737, 8738, 8739, 8740, 8741, 8742, 8743, 8744, 8745, 8746, 8747, 8748, 8749, 8750, 8751, 8752, 8753, 8754, 8755, 8756, 8757, 8758, 8759, 8760, 8761, 8762, 8763, 8764, 8765, 8766, 8767, 8768, 8769, 8770, 8771, 8772, 8773, 8774, 8775, 8776, 8777, 8778, 8779, 8780, 8781, 8782, 8783, 8784, 8785, 8786, 8787, 8788, 8789, 879

Chapter 1

System Description

The AMI Prototyping Board is an inexpensive high-performance system that is ideal for first time users of microcomputers so that they may gain familiarity with the individual devices and recommended applications circuitry. It should also be attractive for OEMs who are looking for a single board microcomputer to include as part of their overall system.

Figure 1-1 shows the block diagram of the AMI 6800 Prototyping Board.

The Prototyping Board is packaged on a 10-1/2" x 12" card (26.67 x 30.48 cm) and has two 86-pin edge connectors, one for the microprocessing unit (MPU) bus lines, and one for I/O.

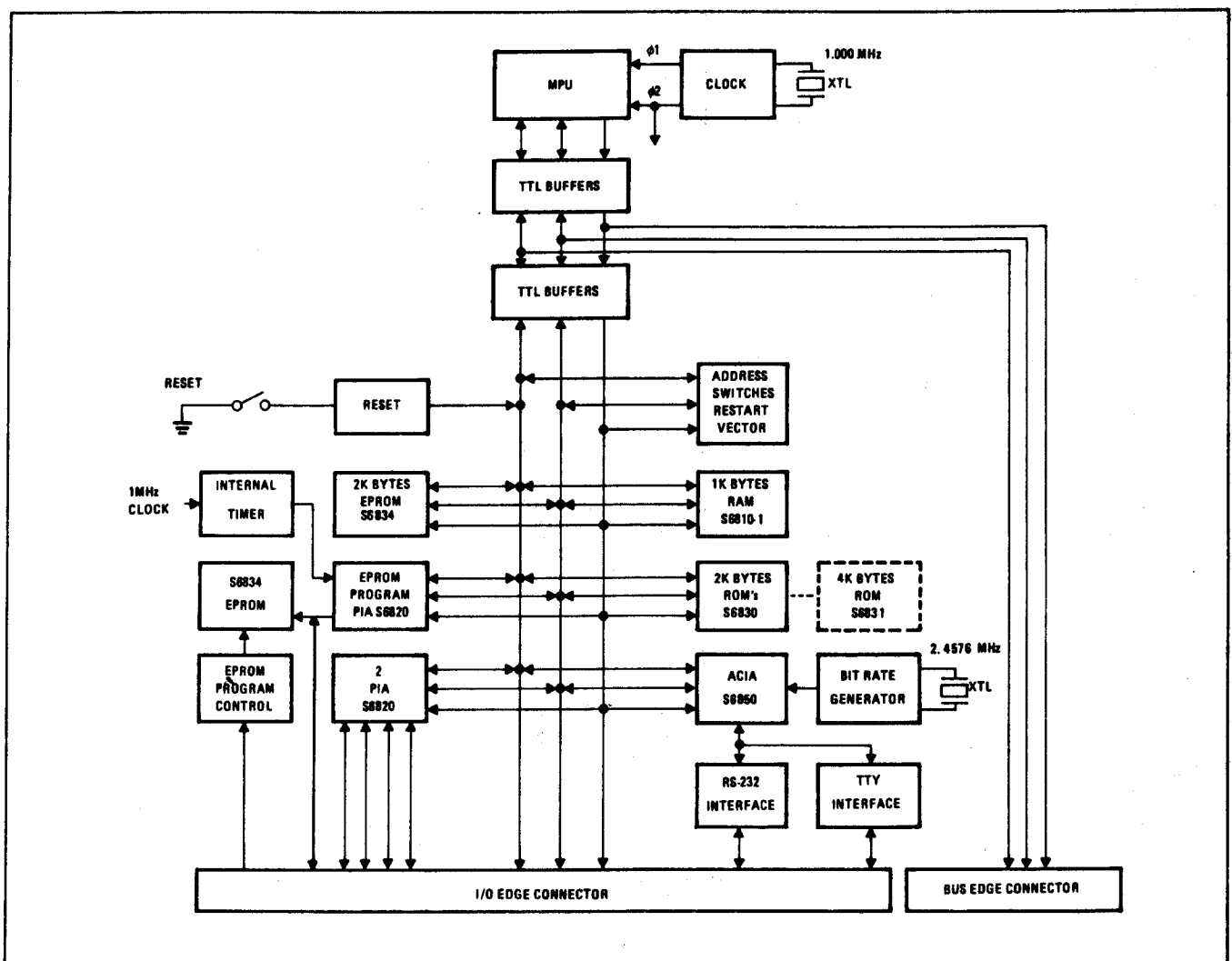


Figure 1-1. AMI 6800 Prototyping Board Block Diagram

MPU

There are three versions of the Prototyping Board. The EVK 300 is a fully built and tested unit with 2K bytes of S6834 EPROM. The EVK 200 is kit version containing all of the components as on the EVK 300 except it has only 512 bytes of EPROM. The EVK 100 is a kit containing only the minimum number of parts to operate with a TTY I/O device.

Central to the Prototyping Board is the S6800 MPU. All data, address and control lines for the MPU are buffered and available at the Bus edge connector. There are also additional circuits for clock generation, reset and restart.

Clock

The basic clock is derived from a 96S02 dual one-shot (IC 12) connected in a regenerative feedback loop. A potentiometer is connected to the RC network of each one-shot which allows the clock to be adjusted from 1.0MHz to approximately 300kHz. The two phases of the clock are driven from 2N5771/5772 transistors. Both phases are buffered and available at the Bus edge connector.

A 1MHz crystal frequency generator is also available for those applications requiring rigid timing accuracies. The 1MHz clock is optionally strapped into the $\phi 1$ one-shot to control the timing accuracy. The 1MHz frequency is also buffered and available on the Bus edge connector.

The clocks can be halted in either $\phi 1$ or $\phi 2$ for cycle-steal, DMA or slow memory applications. Phase 1 is held HIGH by setting the CYCLE STEAL control line LOW. Phase 2 is held HIGH by setting the MEMORY READY line LOW.

The S6800 internal registers are dynamic and it is necessary to refresh them periodically. It is therefore important that the CYCLE STEAL and MEMORY READY lines are not held LOW for more than 5 microseconds. Provisions are made on the Prototyping Board to protect the microprocessor from non-refresh conditions.

Reset

The Reset circuit provides a timed reset for Power On Reset timing and for the Reset switch. The circuit is a timed oscillator which provides a 200 ms reset pulse.

Restart

The starting address of an S6800 is FFFE/FFFF. The contents of these memory locations are put into the Program Counter register each time the MPU is reset. The Evaluation Board traps the FFE/FFFF addresses and puts the contents of the two 8-bit switch sets (IC 32, 43) on the data bus for each address and disabling memory, then gating the first set of switches to the Data Bus during FFFE time and the second set during FFFF time. The user is thus allowed to select any restart address by simply selecting a two byte address on the 16 bits of switch settings.

BUS STRUCTURE

All of the controls to and from the S6800 are available at the Bus edge connector. This allows the user to have complete interface and control to the MPU. TTL buffers are used to isolate MPU lines from the bus. Other buffers are provided to assure that the devices on the board do not violate the maximum load capacity when the board is fully loaded. Bus polarity is the same as on the MPU (data and address true = HIGH).

The bus buffers are non-inverting 3-state hex buffers (8T97). The enable controls to the MPU are always active. Controls for the address bus are gated by the DMA GRANT line. The data bus is controlled by the DMA and R/W lines.

MEMORY ADDRESS ASSIGNMENT

Address assignments have been made such that all components on the card can run in the upper 8K bytes of memory. An address assignment map is shown in Figure 1-2 on page 3. All I/O devices are assigned as shown in Table 1-1, page 3.

Address decoding is made by use of three 74S138 one-of-eight decoders (IC 44, 45, 54). The first decoder (IC 54) selects one 1K-byte block of the upper eight 8K-bytes of memory. The output of this decoder is for RAM, I/O, ROM, or PROM enable lines. The second decoder (IC 44) selects one of eight RAM memory chips. The third (IC 45) selects I/O devices on the board.

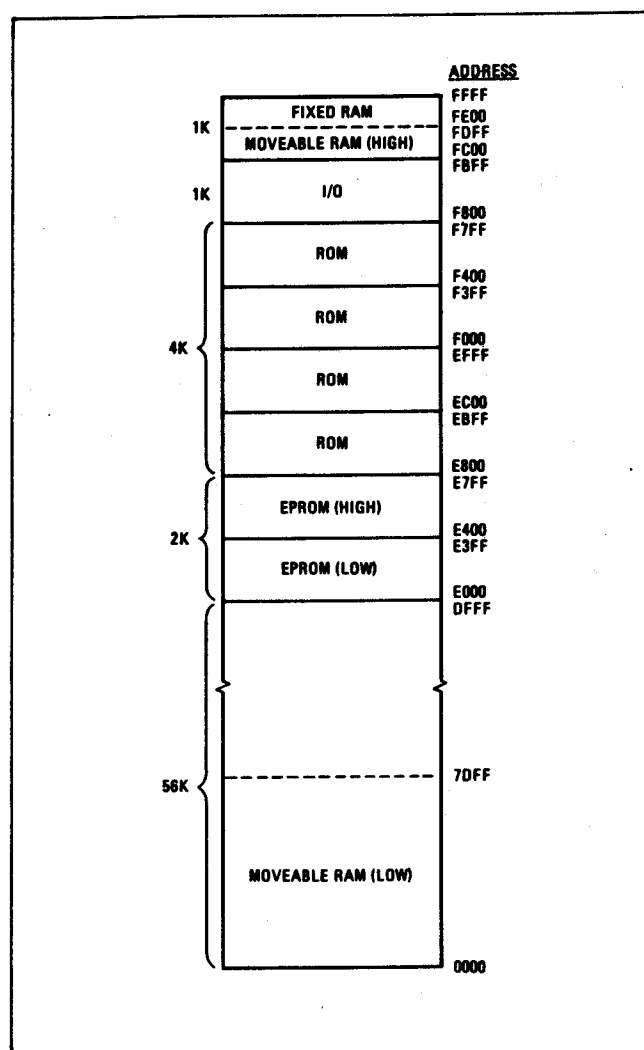


Figure 1-2 Memory Assignment Map for the AMI Prototyping Board

I/O PORT	ADDRESS	ASSIGNMENT
S6850 ACIA	FBCE FBCF	Serial I/O — TTY Status/Read Control/Write
S6820 PIA 1	FBC8 FBC9 FBCA FBCB	Unassigned Peripheral Register A Control Register A Peripheral Register B Control Register B
S6820 PIA 2	FBC0 FBC1 FBC2 FBC3	Keyboard/Unassigned Peripheral Register A Control Register A Peripheral Register B Control Register B
S6830 PIA 3	FBC4 FBC5 FBC6 FBC7	PROM Burner Peripheral Register A Control Register A Peripheral Register B Control Register B

Table 1-1 I/O Address Assignment

A **MEMORY DISABLE** line is available at the Bus edge connector. This line, when LOW, deselects the first address decoder disabling all I/O and memory devices on the board. An **I/O ENABLE** line is derived from the first address decoder and is available at the Bus edge connector. It must be noted that **I/O ENABLE** on the backplane is not valid when **MEMORY DISABLE** is LOW.

READ ONLY MEMORY

The Prototyping Board has assigned locations for two 1K byte S6830 ROMs and for four 512 x 8 S6834 EPROMs. The ROM circuits are designed such that the locations will also accept two 2K byte 16K ROMs (S6831). Thus, maximum memory allocation for ROM and EPROM is 6K bytes. The prototyping operating system program (PROTO) is assigned to the ROM with a starting address of F000.

The four EPROM locations may contain any user program. Execution can start from beginning EPROM location either by selecting EPROM starting address of E000 in the restart switches or by branching to that address using the "G" command in the PROTO program.

RANDOM ACCESS MEMORY

The RAM is divided into two parts, 512 bytes fixed in the highest memory locations and 512 bytes of moveable memory.

Since the highest memory locations (FFFE, FFFF) are used for restart address, the address circuits disable the RAM using a memory disable line and force the 16 bit switch address on the data bus whenever a Reset occurs. This allows the user to vector to any address as his restart address.

The PROTO program assigns restart vectors for IRQ, NMI and SWI whenever it is started (usually via Reset). It is therefore important to note that the user program must do the same thing if he does not use PROTO and restarts from a power down mode.

The stack pointer is assigned to address FF8F in PROTO. This allows the remaining RAM to be used as stack if so desired.

A switch option allows 512 bytes of RAM to be relocatable. When in the upper portion of memory,

the RAM is assigned to addresses FC00 to FDFE making all 1K-bytes of RAM on the board contiguous (FC00 to FFFF). When in the lower portion of memory, the 512 bytes are addressed whenever A9 and A15 are not true (0000 — 01FF for example). It is thus recommended that RAM be assigned to the low address only if the user does not add other RAM to his development system.

PARALLEL I/O

Three S6820 PIA's give the user a wide range of I/O flexibility. The PIA's are assigned addresses as shown in Table 1-1. Interface pins of these devices are directly connected to the I/O edge connector. The CA2 pin for the PIA at addresses FBC4 is also connected to the V_{PROG} input (pin 11) to the EPROM socket (IC 46) through a +5V to -50V driver. The user is cautioned to use this line such that it will not interfere with his I/O function if programming an EPROM. For example, if the CA2 line is connected to an external control function, this function may be erroneously activated while programming an EPROM.

SERIAL I/O

SW POSITION				BIT RATE
4	3	2	1	
0	0	0	0	19,200 baud
0	0	0	1	0 baud
0	0	1	0	50 baud
0	0	1	1	75 baud
0	1	0	0	134.5 baud
0	1	0	1	200 baud
0	1	1	0	600 baud
0	1	1	1	2,400 baud
1	0	0	0	9,600 baud
1	0	0	1	4,800 baud
1	0	1	0	1,800 baud
1	0	1	1	1,200 baud
1	1	0	0	2,400 baud
1	1	0	1	300 baud
1	1	1	0	150 baud
1	1	1	1	110 baud

Table 1-2. Bit Rate Generator Switch Settings,
0 = Closed, 1 = Open

One S6850 ACIA allows the system to communicate bi-directionally with serial data I/O peripherals such as a TTY. A baud rate generator generates all standard communication frequencies by switch selection. This frequency operates independently of the system clock so the MPU frequency can be changed without altering the I/O clock rate. See Table 1-2 for switch setting and associated fre-

quencies. A 20 mA current loop interface and an RS-232 interface are both available at the I/O edge connector.

Address assignments for the ACIA are given in Table 1-1.

EPROM PROGRAMMER

A unique feature of the Prototyping Board is its ability to program AMI S6834 EPROMs. A third PIA latches the address and data information for programming the EPROM. The EPROM socket programs only the S6834 EPROM, however, an adapter plug is available to also program the AMI S5204A EPROM. Except for the V_{PROG} input, all address, chip select, R/W and data I/O pins on both EPROMs are completely TTL compatible and are driven directly from the PIA outputs. The outputs are also available on the I/O edge connector for convenience in using another EPROM programming socket.

Programming is achieved by pulsing the V_{PROG} pin with -50 volts through the CA2 line of the PIA at address FBC4. This line drives the transistor that gates the -50 volt source to the V_{PROG} pin. The -50 volt source is switched ON or OFF via the V_{PROG} switch.

INTERVAL TIMER

The 1MHz clock is divided into two basic interval timing pulses, one every 100μs and one every millisecond, by using three 74160 divide-by-ten counters (IC 50, 51, 52). The time intervals may be changed only by rerouting the printed circuit lands. These basic times are available to the user and are also used for EPROM programming. The 1 ms timer sets bit 7 of address FBC5 and the 100μs timer sets bit 7 of address FBC7.

DMA

Three types of DAM implementation are possible on the Prototyping Board, a halt processor mode, a cycle steal mode and a multiplex mode. A switch selects these DMA modes. The switch must be in the DMA position for the multiplex DMA mode. A delayed clock gives the DMA GRANT line to the bus after the "Data Hold" time has passed for a multiplexed type of DMA operation. The control lines for the halt processor and cycle steal modes are available at the Bus edge connector.

Chapter 2

Physical Description

The AMI 6800 Prototyping Board is designed on a single 10½"X12" printed circuit board. There are two edge connectors, one is used for the MPU bus and the other for the input and output. Voltage assignments are made such that the card may be reversed in a dual connector application without destroying the circuits. The connectors are 43X2 or 86 pins each and fit an Amphenol P/N 225-805-43 connector or equivalent. Table 2-1 lists the I/O pin assignments.

It is possible to operate the Prototyping Board by using only one connector. All power, ground and I/O connect circuits are present in the I/O connector so this is all that is essential to connect in a minimum system. The Bus connector is for expanding the system to add more memory, I/O or controls.

Nominal power requirements are 3.5 amps of +5 volts and 150mA of -12 volts when S6834 EPROMs are used.

Connector	Pin	Assignment	Comments	Connector	Pin	Assignment	Comments
A (MPU Bus)	1	Ground		A (cont'd)	26	D ₄	
	2	Ground			27	A ₅	
	3	Ground			28	D ₅	
	4	Ground			29	A ₆	
	5	+5 volts			30	D ₆	
	6	+5 volts			31	A ₇	
	7	+5 volts			32	D ₇	
	8	+5 volts			33	A ₈	
	9	+12 volts			34	Reserved	
	10	+12 volts			35	A ₉	
	11	+15 volts			36	Reserved	
	12	+15 volts			37	A ₁₀	
	13	-12 volts			38	Reserved	
	14	-12 volts			39	A ₁₁	
	15	-15 volts			40	Reserved	
	16	-15 volts			41	A ₁₂	
A	17	A ₀			42	Reserved	
	18	D ₀			43	A ₁₃	
	19	A ₁			44	Reserved	
	20	D ₁			45	A ₁₄	
	21	A ₂			46	Reserved	
	22	D ₂			47	A ₁₅	
	23	A ₃			48	Reserved	
	24	D ₃			49	(A ₁₆)	
	25	A ₄			50	Reserved	

Table 2-1. I/O Pin Assignments

Connector	Pin	Assignment	Comments	Connector	Pin	Assignment	Comments
A (cont'd)	51	<u>VMA</u>		B (I/O)	1	-50 volts	
	52	Reserved			2	V PROG	
	53	<u>ENABLE</u>			3	EPROM D ₀	
	54	Reserved			4	EPROM A ₀	
	55	<u>MEMORY DISABLE</u>	Open Collector Input		5	EPROM D ₁	
	56	Reserved			6	EPROM A ₁	
	57	MEMORY READY			7	EPROM D ₂	
	58	Reserved			8	EPROM A ₂	
	59	R/W			9	EPROM D ₃	
	60	Reserved			10	EPROM A ₃	
	61	<u>HALT</u>	O.C. Input		11	EPROM D ₄	
	62	Reserved			12	EPROM A ₄	
	63	<u>I/O SELECT</u>	Output		13	EPROM D ₅	
	64	Reserved			14	EPROM A ₅	
	65	<u>RESET</u>	Output/ O.C. Input		15	EPROM D ₆	
	66	16 X Baud Rate			16	EPROM A ₆	
	67	<u>RESET SW</u>	Input		17	EPROM D ₇	
	68	Reserved			18	EPROM A ₇	
	69	<u>RES. SW DISABLE</u>	O.C. Input		19	EPROM R/W	
	70	<u>TSC</u>			20	EPROM A ₈	
	71	1.000MHz Clock			21	CB ₂₋₂	
	72	Bus ϕ 1 Clock			22	CB ₂₋₁	
	73	2.4576MHz Clock			23	CB ₁₋₂	
	74	Bus ϕ 2 Clock			24	CB ₁₋₁	
	75	<u>IRQ</u>	O.C. Input		25	PB ₇₋₂	
	76	BA			26	PB ₇₋₁	
	77	<u>NMI</u>	O.C. Input		27	PB ₆₋₂	
	78	<u>REFRESH GRANT</u>			28	PB ₆₋₁	
	79	<u>CYCLE STEAL</u>			29	PB ₅₋₂	
	80	<u>DELAYED DMA GRANT</u>			30	PB ₅₋₁	
	81	<u>DMA GRANT OUT</u>			31	PB ₄₋₂	
	82	<u>DMA GRANT IN</u>			32	PB ₄₋₁	
	83	<u>INTP OUT</u>			33	PB ₃₋₂	
	84	<u>INTP IN</u>			34	PB ₃₋₁	
	85	Reserved			35	PB ₂₋₂	
	86	-50 volts			36	PB ₂₋₁	
A					37	PB ₁₋₂	
					38	PB ₁₋₁	
					39	PB ₀₋₂	
					40	PB ₀₋₁	
					41	PA ₇₋₂	
					42	PA ₇₋₁	
					43	PA ₆₋₂	
					44	PA ₆₋₁	
					45	PA ₅₋₂	

Table 2-1. I/O Pin Assignments (Continued)

Connector	Pin	Assignment	Comments	Connector	Pin	Assignment	Comments
B	46	PA ₅₋₁		B	66	TTY OUT -	
(cont'd)	47	PA ₄₋₂		(cont'd)	67	$\overline{\text{TXD}}$	
	48	PA ₄₋₁			68	TTY IN +	
	49	PA ₃₋₂			69	$\overline{\text{RXD}}$	
	50	PA ₃₋₁			70	TTY IN -	
	51	PA ₂₋₂			71	-15 volts	
	52	PA ₂₋₁			72	-15 volts	
	53	PA ₁₋₂			73	-12 volts	
	54	PA ₁₋₁			74	-12 volts	
	55	PA ₀₋₂			75	+15 volts	
	56	PA ₀₋₁			76	+15 volts	
	57	CA ₁₋₂			77	+12 volts	
	58	CA ₁₋₁			78	+12 volts	
	59	CA ₂₋₂			79	+5 volts	
	60	CA ₂₋₁			80	+5 volts	
	61	RTS			81	+5 volts	
	62	READER CONTROL			82	+5 volts	
	63	CTS			83	Ground	
	64	TTY OUT +			84	Ground	
B	65	DCD		B	85	Ground	
					86	Ground	

Table 2-1. I/O Pin Assignments (Continued)

ANN.

Comments	Assignment	Connector Pin	Comments	Assignment	Connector Pin
	TTY OUT	44			44
	TTY	45			45
	TTY IN	46			46
	TXD	47			47
	TTY IN	48			48
	TXD	49			49
	TTY IN	50			50
	-15 volts	51			51
	-15 volts	52			52
	-15 volts	53			53
	-15 volts	54			54
	+15 volts	55			55
	+15 volts	56			56
	+15 volts	57			57
	+15 volts	58			58
	+5 volts	59			59
	+5 volts	60			60
	+5 volts	61			61
	+5 volts	62			62
	Ground	63			63
	Ground	64			64
	Ground	65			65
	Ground	66			66

Table 2-1. I/O Pin Assignments (Continued)

Chapter 3

Software

The board will be supplied with a prototyping operating system program (PROTO). The program resides in ROM with a starting address of F000. The various routines within PROTO are called by entering via the TTY keyboard one of the commands described in the following paragraphs. A command consists of one character command identifier followed by additional parameters, if needed, separated by blanks or commas. All commands end with a carriage return. Since no action is taken before the carriage return, an input line may be deleted by the use of the TTY ESCAPE key.

L, ADDL, ADDH, OFFSET

The Load tape command loads data from a hex formatted tape (see Appendix A) into the user's memory between ADDL and ADDH, inclusive. The OFFSET is added to the memory address specified on the tape to form the actual memory starting address for the data stored. If a byte to be stored into memory has an address outside of the range ADDL, ADDH, it is not entered into memory, but a Delete character (H'FF) is transmitted to the terminal.

Example: L 0100 02FF FFFA

The address range in the L command is optional, and if omitted is assumed to be the full range of memory (0000—FFFF). The offset parameter is also optional, and if omitted is assumed to be zero (0000). Thus the L command with no parameters loads the tape into the memory locations specified on the tape with no offset. The offset value in the L command is a two's complement signed number, entered in unsigned hexadecimal. For example, an offset of -6 is entered as FFFA.

If an attempt is made to load non-existent memory, or ROM, the loading operation will terminate, typing out the address and the message "BAD ADR".

In operating the Load command, PROTO turns on the tape reader and scans the tape for the first ASCII "S", which indicates start of record. It is not necessary to position the tape at the first record of a tape file since each record contains its own starting address.

PROTO will load data records until it encounters an end of file (EOF) record or a tape error (Check Sum or illegal character). When PROTO reads a header record (start of record and address), it translates the header into ASCII characters and prints the result. The Check Sum is the binary sum of all characters in the block.

PROTO does not list the tape contents as the tape is being read.

When PROTO encounters an end of file record or a tape error, it turns off the reader and prints "EOF" or "CKSM ERR" respectively.

P, ADDL, ADDH, OFFSET

The Punch hex format command causes PROTO to punch on the TTY papertape the contents of memory between ADDL and ADDH, inclusive. Each record is punched with a four-digit hex address of the starting byte of the record. This address is derived from the memory address of the byte being punched, plus the offset value, OFFSET. The offset is optional, and if omitted is assumed to be zero.



All data records are punched in hex format. Records using this command (except the last record) contain 16 bytes of data plus the start code, byte count, address, and the checksum.

The P command does not cause an EOF record to be punched so that several disjoint blocks of memory can be combined on one tape file.

Example: P F000 F07F 0F00

S, ADDR, BYTE1, BYTE2, — — —, BYTEN

The Set memory command writes the 8-bit data words specified by BYTE1 to BYTEN into consecutive memory locations starting at ADDR.

If ADDR has more than 4 (hexadecimal) characters or if any of the data bytes have more than 2 characters each, only the last 4 or 2 characters are used respectively.

Example: S 0000 86 05 97 28

Memory locations at 0000 thru 0003 are loaded as shown.

D, ADDL, ADDH

The Display memory command prints the contents of memory between ADDL and ADDH, inclusive, in hex format. Up to sixteen bytes per line are printed, preceded by the hexadecimal address of the first byte of the line. A carriage return is forced after a byte having a low order digit of F in its memory address is printed.

Example: D FC00 FC1F

Two lines of memory contents are printed as follows:

```
FC00 00 01 02 03 04 ... 0E 0F
FC10 10 11 12 13 14 ... 1E 1F
```

G, ADDR

The Go command starts execution of the user program at the address specified by the input parameter. To insure that all registers contain the same information they held before the user program was interrupted, PROTO pushes into the stack the copy of the user registers that it keeps at locations FFEB—FFF3 (CC, B, A, X, P, S) then executes an RTI instruction. The user can change

the initial values of the registers by changing the contents of these locations.

Example: G 300

Program will branch to address 0300 and start execution from that point.

R

The Registers command prints the contents of memory locations FFEF—FFF3 which contain the values that were in the user's C, B, A, X, P, and S registers (in that order) when the user's program was last interrupted.

B, ADDL, ADDH, ROMAD

The Burn command copies the contents of user memory into the EPROM in the programming socket, beginning with memory location ADDL through ADDH, inclusive, to EPROM locations beginning with address ROMAD. Each byte is burned in with 20 3-ms pulses of -50V on the VPROG pin (pin 11) of the EPROM. Before attempting to write into the EPROM, the contents of the EPROM are compared with the user memory data byte to verify that the EPROM will take the byte (PROTO will not attempt to program a EPROM location to logic LOW which already contains logic HIGH). After the 20 pulses, the new contents of the EPROM are verified against the memory byte to be sure the data was indeed written. If the byte did not program, a NAK code is typed out on the terminal, and another try is made, up to a maximum of three tries.

If the preverify encounters a EPROM location containing HIGHs where the memory byte has zeros, PROTO will type out the memory address, the memory byte in binary, the EPROM byte in binary, and the EPROM address (if different from the memory address), then stop. If after attempting to write data into the EPROM, the data does not program, or erroneous bits show up, a similar display occurs for the failing location, with the additional message "BAD ADR" typed on the same line.

The EPROM address ROMAD is optional, and if omitted, ADDL is used, with only the least significant nine bits of the address being used. If the address range ADDL, ADDH is omitted, the 512 bytes beginning at FC00 are used, and the EPROM

is checked to insure it contains all LOWs before any locations are written. If not, four question marks are typed and the B command is aborted.

V, ADDL, ADDH, ROMAD

The Verify command compares user memory between ADDL and ADDH, inclusive, with the corresponding locations in the EPROM in the programming socket, beginning with EPROM address ROMAD. Each location that does not match is typed out in the following format:

```
aaaa mmmmmmmm pppppppp rrrr
```

where "aaaa" represents the user memory address, "mmmmmmmm" represents the memory byte, in binary, and "rrrr" represents the EPROM address, if different from the memory address (in the low nine bits). Nothing is typed for matching locations. The typeout may be aborted by typing an ESC key during the typeout.

If the ROMAD parameter is omitted, ADDL is assumed. If no parameters are supplied in the command, the whole EPROM is compared to the contents of FC00 — FDFF.

I, ADDL, ADDH, ROMAD

The Input command copies the contents of an EPROM in the programming socket into memory beginning at the address ADDL through ADDH, inclusive, from the EPROM address ROMAD. If ROMAD is omitted, ADDL is assumed. If no parameters are supplied, the entire EPROM is copied into the RAM area, FC00 — FDFF. An attempt to copy an EPROM into non-existent memory will abort the command with the message "BAD ADR".

M, ADDL, ADDH, DEST

The Move command copies memory from the range ADDL — ADDH, inclusive, to the RAM locations starting at DEST. This copy begins at the lower address, so if DEST lies within the range ADDL — ADDH, some of the original data will be lost, and other parts will be duplicated.

E

The End of Transmission command is used to cause an EOT character to be punched on the paper tape. After a field has been punched, an EOT will terminate the record and punch a trailer tape. When reading a record, the reader will stop at the

EOT character. If no EOT character is present, the reader must be manually turned off and the Reset switch must be pressed to enter the operating system program.

THE SUBROUTINE ROM

Many of the monitor's functions are accomplished with the help of the Re-Entrant Self-Relative Subroutine ROMs (RS)³. This standard ROM, which can be considered a software extension to the 6800 instruction set, is also available to be used by the user both on the prototype board and in his final production system. The user can call one of the 25 (RS)³ subroutines with an SWI instruction followed by the number of the desired subroutine. The details of the subroutines available in the (RS)³ user's manual.

The user should be aware of the fact that the (RS)³ pushes from 7 to 10 bytes of data onto the stack, depending upon which subroutines are called. This means that if the user calls (RS)³ routines, he must make sure that the necessary memory space is available for stack expansion.

Since PROTO assigns its own stack area, the user need not be concerned about how (RS)³ is used.

INTERRUPTS

Of the four available interrupt vectors, IRQ, RESET and SWI are used by PROTO while NMI is left for the user. The vectors are in RAM (except for RESET which is switch controlled) so the user writing his own program can completely control the system.

The upper memory locations are RAM. If the user expects either NMI or IRQ interrupts to occur, he must initialize the vector addresses to the starting address of the IRQ and NMI handler routines.

PROTO must have control of the RESET vector so that the RESET switch on the Prototyping Board can return program control to PROTO at any time.

The reset routine copies the contents of the B, A, X, CC, and S registers into a fixed area of memory. This means that the program can be aborted at any time by using the reset switch while still saving all the registers except the program counter. Unfortunately, the contents of the program counter are lost.

It is possible for the user to use the NMI interrupt to abort a program execution without losing the contents of the P and C registers. This condition is automatically set in the NMI handling routine when PROTO is called. This interrupt vector will cause the contents of the user's registers to be printed when the NMI lines goes low.

Since the SWI instruction is used to call subroutines between 00 and H'18 from (RS)³ as described in Chapter 3 "The Subroutine ROM", the user is somewhat limited in the ways he can use SWI instructions. However, he can access an SWI handler routine in his own program by an SWI instruction followed by a byte containing the decimal number less than H'80 but greater than H'19 < n < H'80 sequence, PROTO passes control at address FFF4. If the user expects to access his own SWI routine and use PROTO, he must use the Set Memory command to store the address of this routine at locations FFF4 and FFF5.

PROTO makes sure that the user's SWI routine is entered from the stack with all registers containing the same information that they would hold if the routine were entered directly through the SWI vector.

BREAKPOINTS

Breakpoints allow the user to halt his program and examine the contents of the internal registers. PROTO provides two types of breakpoints. In this system, breakpoints are actually debugging routines that can be called from the user's program just like

(RS)³ routines. (See Chapter 3 "The Subroutine ROM".)

Each breakpoint requires a two byte calling sequence: an SWI instruction followed by a number.

Breakpoints may be inserted either by reassembling the program with the extra SWI instructions added the Set Memory command may be used to replace parts of the code with SWI instructions. Note that the second method is not satisfactory for the snapshot option (described below) since the replaced code must be restored before execution can be continued. When using the second method, the user must make sure that he replaces the first two bytes of an instruction. If the SWI replaces the second or third byte of an instruction, it may be interpreted as an address rather than an opcode.

The different types of breakpoints are:

1. Print registers (SWI, H'80)
2. Snapshot (SWI, H'81)

The sequence SWI, H'80 saves the user's registers at the vector stored in FFF4 — FFF5, prints their contents (in the order CC BB AA XXXX PPPP SSSS), then returns control to PROTO.

The sequence SWI, H'81 prints out the contents of the user's registers then continues executing the user's program starting at the address following the byte containing the number H'81. Note that if this address does not contain a valid opcode, unpredictable results will occur.

Chapter 4

Reentrant Self-Relative Subroutine ROMs (RSRSRS) = (RS)³

The cost of microprocessor software development is many small items: the cost of assembly time, storage time, transmission time, loading time, design, development, documentation and debug. The cost of many of these items continues to accumulate even though a subroutine library exists for common functions, in particular the time and cost of transmission, loading and ROM pattern generation.

The purpose of Reentrant Self-Relative Subroutine ROMs (RS)³ is to give the user a hardware subroutine package which exists in the breadboard design from the beginning. The programs are documented, debugged and constitute some of the most commonly performed subroutines that assembly language programmers generate. The subroutines are not complex and are not intended to be. Any subroutine could easily be reproduced by a user; however, the intention is that the routine exists now and the user does not have to reproduce it. The routines tend to be slow because of their generality but the intention is immediate availability. If a particular program is time critical, it can be regenerated later when the time critical elements are known.

CONCEPTS

The (RS)³ uses a number of concepts to allow flexibility in the user environment. The first concept is self-relative programming. This simply means that the program will function correctly regardless of where it is located in memory. The user will need to know where it is located so he can reference it. However, this actual location will only have to be recorded once. The self-relative program uses relative address instructions for program control and the index and stack pointer instructions for data manipulation.

The stack is used for temporary storage of data to prevent (RS)³ from being tied to fixed addresses. This allows the program to be reentrant; i.e. the program can be called at different times without completing the previous call. This means that the same routine can be called by the interrupt processor as well as by the program which was interrupted. The concept of reentrant code is not to be confused with recursive code; even through recursive coding could have been used in the subroutine package, it is not.

The subroutine calling mechanism uses the SWI instruction followed by a single byte index for the particular subroutine invoked. This was chosen because the SWI is the most convenient from an internal programming viewpoint and the safest. It is safe because an error in a ROM can be corrected by replacing the subroutine ROM without altering any other user ROM. If direct addresses to subroutine code exist in the user's domain, his ROMs would change if the location of the routine in the (RS)³ changed.

IMPLEMENTATION

The user places the base address of the (RS)³ into the SWI vector address. Each SWI instruction requires an index byte to follow the SWI instruction where the index indicates the function to be executed. After the function is performed, the user program will continue with the instruction following the index byte. In essence, a whole new set of instructions have been created for the user which are two bytes long.

To make the entry easier, a macro call can be provided which will assemble the correct index byte when the function name is used. A set of EQU assembler commands associates the name and the index byte value.



Example:

MUL8	EQU	10
MUL16	EQU	11
DIV8	EQU	12
DIV16	EQU	13
	.	
	.	
FUN	MACRO	INDEX
	SWI	
	BYTE	INDEX
	MEND	
	.	
	.	
	FUN	MUL8

Each (RS)³ ROM will have the ability to interrogate the index byte and vector to the appropriate subroutine if it is included in the ROM. If the index extends the number of subroutines included on the ROM the number is subtracted from the temporary index value and the next (RS)³ ROM is automatically branched to. This allows the user to select any of several subroutine sets, where each set of subroutines is represented by a separate ROM. The selected ROMs are concatenated together into a contiguous region of the user's memory space, and are automatically linked together by the index value. Thus the actual value of the index byte for any particular subroutine is the sum of the total number of subroutines in the physically previous (RS)³ ROMs plus the offset in its own ROM. The document for each (RS)³ ROM will therefore include both the descriptions of the subroutines in that ROM and the relative index values for these subroutines. It must be noted that address assignments for (RS)³ ROMs must be made beginning at 1K boundary addresses.

THE PROTO SYSTEM

The 2K \times 8 ROM provided with the PROTO prototyping system includes a set of (RS)³ subroutines with a slightly different linkage from the standard (RS)³ form, although the calling sequence is the same. In particular, the provision for additional subroutines in the form of other (RS)³ ROMs is limited to a total of 127 subroutines. The first additional (RS)³ ROM address must be placed in RAM location FFF4 (which can be set via the Set Memory command or modified by an initialization code in a user program). Also, since it is incorporated into a larger program, the whole of which very nearly fills the 2K bytes of its ROM, the

(RS)³ part of the ROM does not start on an even page boundary, making it awkward for isolated use. However, the 24 subroutines included in this ROM are available to user program calls with the SWI calling sequence, as described.

(RS)³ IN PROTO – SUBROUTINE DESCRIPTIONS

Each of the subroutines in the ROM are described here, giving the index for the call, a mnemonic subroutine name, a descriptive title, and the number of bytes in the stack used by the call (including the SWI). A brief description of the subroutine operation is also given, with the entry requirements, the exit conditions, and the registers altered by the subroutine. Only those registers indicated are altered by any (RS)³ subroutine.

Index	Name	Title	Stack Bytes
00	PUSH ALL	Push All Registers	14
<p>Five bytes are pushed onto the stack, containing, respectively, the Condition Codes, the B and A accumulators, and the Index Register. No registers are altered (except the stack pointer, which is decremented by 5).</p> <p>Entry: Any</p> <p>Exit: Stack: SP +1 +2 +3 +4 +5 (=old SP) CC, B, A, XHXL</p> <p>Registers Altered: SP</p>			
01	POPALL	Pop (=Pull) All Registers	9
<p>Five bytes are pulled from the stack into the Condition Codes, the B and A accumulators, and the Index register, respectively. The Stack Pointer is incremented by 5.</p> <p>Entry: Stack, as by PUSH ALL</p> <p>Exit: CC, B, A, X pulled from stack</p> <p>Registers Altered: CC, B, A, X, SP</p>			
02	TXAB	Transfer Index Register to A and B	9

The most significant eight bits of the index register are copied to the A accumulator, and the least significant eight bits are copied to the B accumulator.

Entry: Any
Exit: A, B loaded from X
Registers Altered: A, B

Index	Name	Title	Stack Bytes
-------	------	-------	-------------

03	TABX	Transfer A and B to Index	9
----	------	---------------------------	---

Accumulator A is copied to the most significant byte position of the index register, and accumulator B is copied to the least significant byte position of the index register.

Entry: Any

Exit: X loaded from A, B

Registers Altered: X

04	XABX	Exchange A and B with Index	12
----	------	-----------------------------	----

The contents of the Index register and the two accumulators are exchanged, A with the most significant byte of X, B with the least significant byte.

Entry: Any

Exit: A, B and X exchanged

Registers Altered: A, B, X

05	PUSHX	Push Index Register	11
----	-------	---------------------	----

The contents of the index register is pushed onto the stack. The Stack Pointer is decremented by two.

Entry: Any

Exit: Stack: SP +1 +2 (=old SP)
XH XL

Registers Altered: SP

06	PULLX	Pop (=Pull) Index Register from Stack	9
----	-------	---------------------------------------	---

Two bytes are pulled from the stack into the index register, and the stack pointer is incremented by two.

Entry: Two bytes on stack

Exit: X pulled from stack

Registers Altered: X, SP

Index	Name	Title	Stack Bytes
-------	------	-------	-------------

07	ADDXAB	Add Index to A and B	14
----	--------	----------------------	----

Add the contents of the Index Register to the two accumulators, as a 16-bit sum, leaving the result in the two accumulators. The most significant byte is assumed to be in accumulator A. The condition codes are set according to the result.

Entry: Addend in X, augend in A, B

Exit: Sum in A, B

Condition

Codes: H = carry from bit 11 to bit 12 of sum
N = bit 15 of sum
Z = 1 if sum is zero; else = 0
V = 1 if two's complement overflow
C = carry out of bit 15 of sum

Registers Altered: A, B, CC

08	ADDABX	Add A and B to Index Register	9
----	--------	-------------------------------	---

Add the contents of the two accumulators to the Index register, leaving the 16-bit sum in the index register. Accumulator A is assumed to be more significant than accumulator B. The condition codes are set according to the result.

Entry: Addend in A, B; augend in X

Exit: Sum in X

Condition Codes: H = carry from bit 11 to bit 12 of sum
N = bit 15 of sum
Z = 1 if sum is zero, =0 otherwise
V = 1 if two's complement overflow
C = carry out of bit 15 of sum

Registers Altered X, CC

Index	Name	Title	Stack Bytes	Index	Name	Title	Stack Bytes
09	ADDAX	Add A to Index Register	9	0C	SUBABX	Subtract A and B from Index Register	9
Add the A accumulator to the contents of the Index register, and return the sum to the index register. The Condition Codes are set according to the result.				Subtract the contents of the A and B accumulators from the Index register, leaving the difference in the Index. The Condition Codes are set according to the result.			
Entry: Addend in A, augend in X				Entry: Subtrahend in A, B; minuend in X			
Exit: Sum in X				Exit: Difference in X			
Condition Codes: (Same as ADDABX)				Condition Codes: (Same as SUBXAB)			
Registers Altered: X, CC				Registers Altered: X, CC			
0A	ADDBX	Add B to Index Register	9	0D	SUBAX	Subtract A from Index Register	9
Add the contents of the B accumulator to the Index register, and leave the sum in the Index register. The Condition Codes are set according to the result.				Subtract the contents of the A accumulator from the contents of the Index register and return the difference to the index register. The Condition Codes are set according to the result.			
Entry: Addend in B, augend in X				Entry: Subtrahend in A, minuend in X			
Exit: Sum in X				Exit: Difference in X			
Condition Codes: (Same as ADDABX)				Condition Codes: (Same as SUBXAB)			
Registers Altered: X, CC				Registers Altered: X, CC			
0B	SUBXAB	Subtract Index from A, B	14	0E	SUBBX	Subtract B from Index Register	9
Subtract the contents of the index register from accumulators A and B as a 16-bit difference. The Condition Codes are set according to the result.				Subtract the contents of the B accumulator from the Index register, leaving the difference in the index register. The Condition Codes are set according to the result.			
Entry: Subtrahend in X, minuend in A, B				Entry: Subtrahend in B, minuend in X			
Exit: Difference in A, B				Exit: Difference in X			
Condition Codes: ^				Condition Codes: (Same as SUBXAB)			
H = undefined				Registers Altered: X, CC			
N = bit 15 of difference							
Z = 1 if result is zero, =0 otherwise							
V = 1 if two's complement overflow							
C = borrow into bit 15 of difference							
Registers Altered: A, B, CC							

Index	Name	Title	Stack Bytes
0F	P2HEX	Print Byte in Hex	15
The byte pointed to by the address in the Index register is converted to hexadecimal notation in ASCII, and output to the ACIA located as follows: memory locations FFF6—FFF7 contain an address of a pair of bytes (indirect pointer) which in turn contain the address of the ACIA status register.			
	FFF7	iL	
	FFF6	iH	
	...		
	i+1	aL	
	i	aH	
	...		
	a+1	ACIA Data	
	a	ACIA Status	

Each byte of the output is stored into the ACIA data register after bit 1 of the Status register is true. The Control register of the ACIA is not altered, and the Data register is not read by this routine. The Index register is incremented past the byte which is output.

Entry: Memory byte at (X); ACIA at (FFF6)

Exit: (two ASCII bytes output)

Registers Altered: X

10	P4HEX	Print Address in Hex	15
The two bytes in memory pointed to by the Index register are converted to four ASCII digits and output to the ACIA located at the address pointed to by the pointer pointed to by the byte pair at FFF6—FFF7 (see P2HEX). The Index register is incremented by two.			

Entry: Two bytes at (X); ACIA at ((FFF6))

Exit: (four ASCII bytes output)

Registers Altered: X

11	PRINTA	Print the Byte in A	10
The byte in accumulator A is output to the ACIA, the address of whose address is in locations FFF6—FFF7. No registers are altered except the ACIA data register.			
Entry: Character in A			
Exit: (one byte output)			
Registers Altered: None			

Index	Name	Title	Stack Bytes
12	PMSG	Print Message String	12
A message string, the first byte of which is pointed to by the Index register, is output to the ACIA, the address of whose address is in locations FFF6—FFF7. The string is terminated by an ASCII EXT (=hex 04), and the Index register is left pointing to that byte on return.			
Entry: Character string to (X) terminated by 04; ACIA at ((FFF6))			
Exit: (in ASCII bytes output), X pointing to 04 byte			
Registers Altered: X			

13	VALAN	Validate AlphaNumeric	11
The character pointed to by the Index register is analyzed, and the Carry flag is set if it is a letter or digit; if it is not a hexadecimal digit, the Overflow flag is set. Other than the condition codes, no registers are altered.			
Entry: Memory byte (ASCII) at (X)			
Exit: Condition			
Codes:			
H = undefined			
N = undefined			
Z = 0			
V = 0 if character in range 0—9, A—F; else = 1			
C = 1 if character in range 0—9, A—Z; else = 0			

Registers Altered: CC

14	INPUTA	Input ACIA byte to A	9
One byte is input from the ACIA, the address of whose address is at location FFF6—FFF7, and this byte is returned to accumulator A. The ACIA is not written to, and except for the A accumulator, no registers are changed. (RS) ³ samples bit 0 of the status register of the ACIA, and when it goes to one, reads the data register. The input byte has bit 7 removed (set to zero).			
Entry: (one byte input)			
Exit: Character in A, bit 7=0			
Registers Altered: A			

Index	Name	Title	Stack Bytes
15	CONHB	Convert Hex String to Binary	11

A string of characters in memory beginning at the address in the index register is scanned for valid Hexadecimal digits; when one is found, it and all immediately following hex digits are converted to a binary number, which is left in the A and B accumulators (A is more significant). When this routine is called, the maximum length of the string is in the B accumulator. On exit, the Carry flag is set to one if the conversion resulted in a valid binary number, and the index register is left pointing to the next character in the string, or if the string is exhausted before finding any hex digits, to the last character of the string.

Entry: Character string (including ASCII hex number) at (X)
Max string length in B (<128)

Exit: Binary number in A, B

Condition

Codes: H = undefined
N = undefined
Z = undefined
V = undefined
C = 1 if valid number; = 0 if not

Registers Altered: A, B, X, CC

16	INDEX	Multiply A × B and Add to Index	12
----	-------	---------------------------------	----

The contents of the A accumulator is multiplied by the contents of the B accumulator, and the product is added to the Index register. The Condition Codes are set according to the result.

Entry: Multiplicand in A, Multiplier in B, augend in X

Exit: Sum in X

Condition

Codes: (Same as ADDABX)

Index	Name	Title	Stack Bytes
-------	------	-------	-------------

17	MUL8	Multiply A Times B	12
----	------	--------------------	----

Multiply the contents of the A accumulator times the contents of the B accumulator, and leave the product in both accumulators as a 16-bit number, with the most significant part in A. This is an unsigned multiply, and if either or both of the factors is negative (two's complement signed) the product will not be a true signed product of the signed factors, as may be seen in this formula:

$$(-n) \times (m) = (256 - n) \times m = 256m + (-nm)$$

The condition codes are nonetheless set according to the result.

Entry: Multiplicand in A, multiplier in B

Exit: Product in A, B

Condition

Codes: H = undefined
N = bit 15 of product
V = 0
Z = 1 if product is zero;
otherwise = D
C = 0

Registers Altered: A, B, CC

Chapter 5

Operating Procedures

This chapter details the specific requirements of the AMI 6800 Prototyping Board that the operation must meet in order to make the board fully functional. The board requires a power supply and an I/O device such as a TTY terminal in order to be fully operational.

Power can be applied through either or both edge connectors. The edge connector is an Amphenol P/N 225-805-43 or equivalent. The basic board (with TTY current loop interface and no EPROM devices) requires only +5 volts for power. Both +5 and -12 volts are required for any operation using EPROMs and +5 and ± 12 volts are required for RS-232 operations. Table 5-1 lists the current specifications for each voltage.

Voltage	Current	Use
+5V	4.0 Amps	General Logic
-12V	150 mA	EPROM and RS-232
+12V	25 mA	RS-232
-50V	35 mA	EPROM Programming

Table 5-1. Prototyping Board Current Requirements

The Board requires a standard TTY ARS 33 type

terminal or equivalent as all operating programs assume TTY and ASCII protocol. A baud rate switch (see Figure 5-1) is provided for all standard baud rates. Default condition (open or no switch) is 110 baud. See Table 1-2, page 4 for full details on the baud rate selection.

The clock frequency is selected via the Clock switch (see Figure 5-1). Clock source is either from a crystal or a pair of one-shots. If the crystal source is used, the one-shot times must not be shorter than 430 ns for phase 1, and 450 ns for phase 2; the total time cannot exceed 1 μ s.

If a TTY is used through the current loop interface, the following jumper must be installed to tie-up the DCD and RTS input:

Jumper Pins B61 & B63 on the I/O edge connector to +5 volts.

The DMA switch should be down (Halt Processor or Cycle Steal Modes) if no DMA is being used. Use the up position only for systems using Multiplex mode of DMA.

The -50 volt switch should be off unless programming an EPROM.

Four holes are provided in the board for soldering heavy gauge wire directly to the board if so desired. They are marked as to which voltage lead goes to which location.

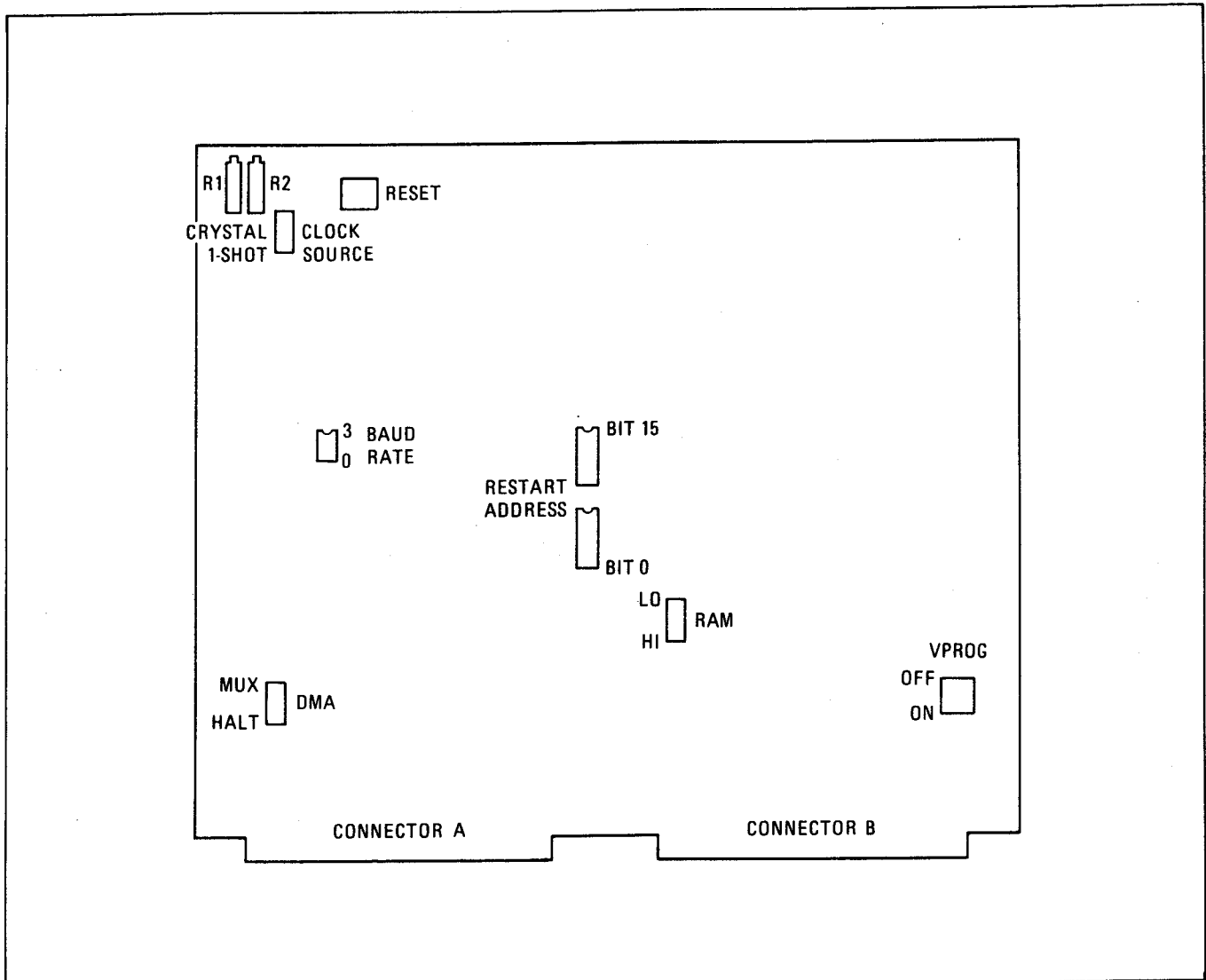


Figure 5-1. AMI 6800 Prototyping Board Switch Configuration

Chapter 6

Troubleshooting Procedures

Although AMI has taken special precautions to ensure reliability of Prototyping Boards by conservative design and component testing and burn-in, occasions may arise where the user may need to troubleshoot a failure. Special handling must be considered for all MOS devices. The following procedures are recommended to test the board for a failure condition.

The first check that should be made is to check for proper 5V distribution throughout the board. This is best done with a digital volt meter, but a scope is adequate. Be sure that the voltage does not drop below about 4.8V at any point on the board, referenced to the local ground plane.

The next section to check in detail is the clock section. It can be tested by observing the oscilloscope photographs that follow and the corresponding reference number on the schematics. The pin numbers for scoping are referenced on the bottom of each photograph. For all operations in this section, it is assumed that the switch if it is installed is set for single-shot regenerative triggering and, if it is not installed, the jumper is set for single-shot regenerative triggering.

Once the clock section is working the other sections can be tested in the following order.

- 1) Reset
- 2) Interval Timer
- 3) Restart
- 4) CPU Section
- 5) Memory and I/O
- 6) Serial and Parallel I/O Operation
- 7) PROM Burner Section

The accompanying photographs should be of great

help in troubleshooting the board. In the following photographs, if more than one trace is present, the second trace shows the 0V reference level. There were several points on the board that were difficult to photograph due to the non-repetitive nature of the wave form. To observe these wave forms, the best procedure is press the Reset button repeatedly and observe the wave form upon release. The operation of the Restart circuits, ACIA selection and ACIA output may be observed in this manner.

The majority of the pictures in the following sequence were taken with the Prototype Board in an idle state waiting for an input from the TTY. Pictures 18, 19 and 21 were taken using the Reset button.

Picture 23, which is the Memory disable pin, is also the one activated by the Restart circuitry. This is normally a one shot operation following depressing of the Reset button. For the purposes of this picture, the processor was set in a small short loop where it was just loading the index from FFFE and FFFF to show the wave form. It will normally not repeat that quickly.

Pictures 26 and 27 show the PROM burner operation and the relative wave forms at 10V per cm vertically, with the second line being the reference line in the photograph. These photographs should be very useful in sorting out any difficulties with the board.

In the event a Prototyping Board does not function when power is applied and after all troubleshooting directions have been followed without success, the board can be returned directly to AMI for repair. The charge for repair by AMI for an EVK 300 Kit is \$125.00 after the warranty has

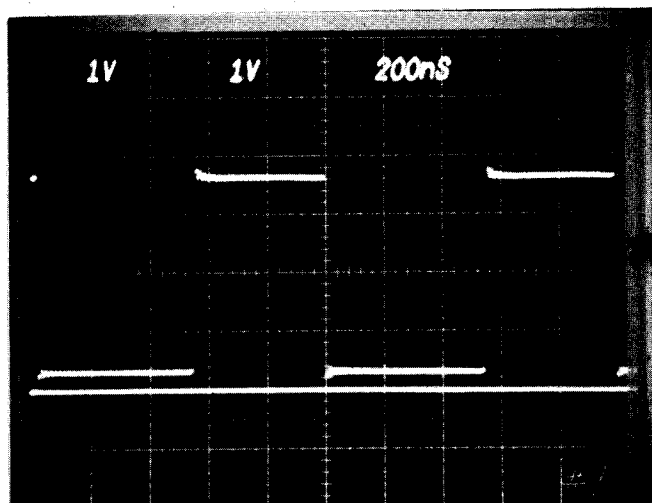


expired. (In the event of obvious user gross negligence, higher repair charges may be quoted.)

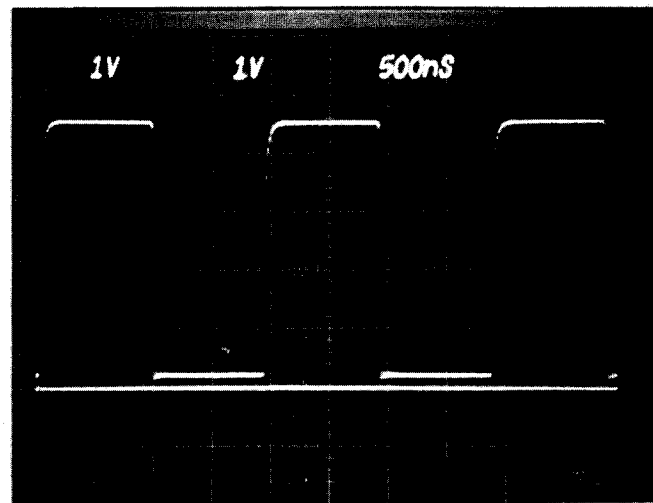
Repack the board in the shipping box and mail it to:

EVK Service
c/o American Microsystems, Inc.
3800 Homestead Road
Santa Clara, CA 95051

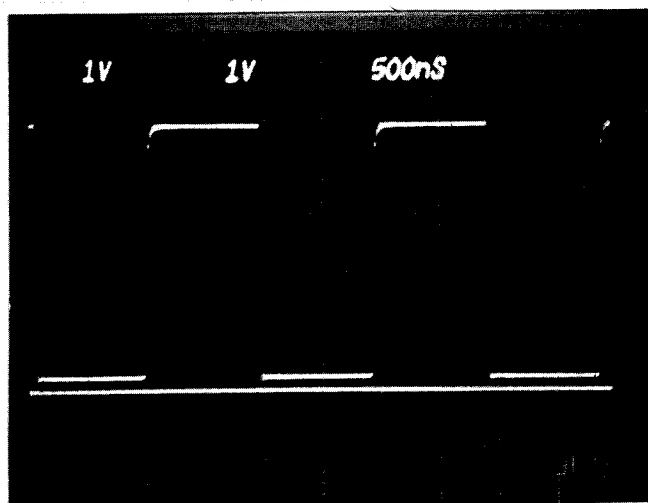
Enclose a check for the amount specified payable to AMI. AMI will repair the board and return it. Be sure your name and mailing address is enclosed so that the board can be promptly returned. An indication of the failure mode would also be of great assistance.



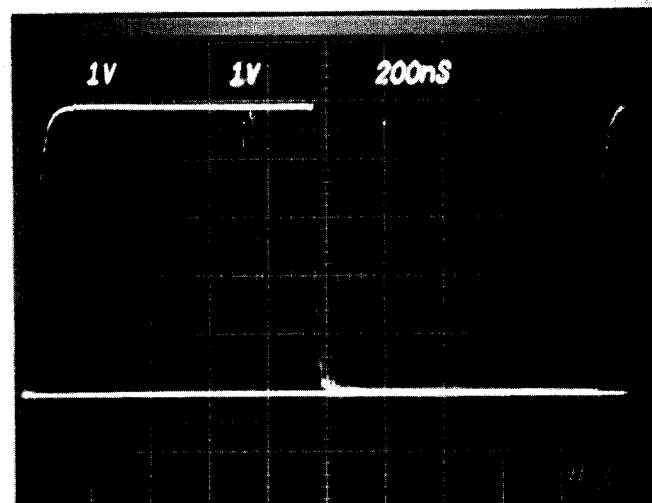
D14-P6



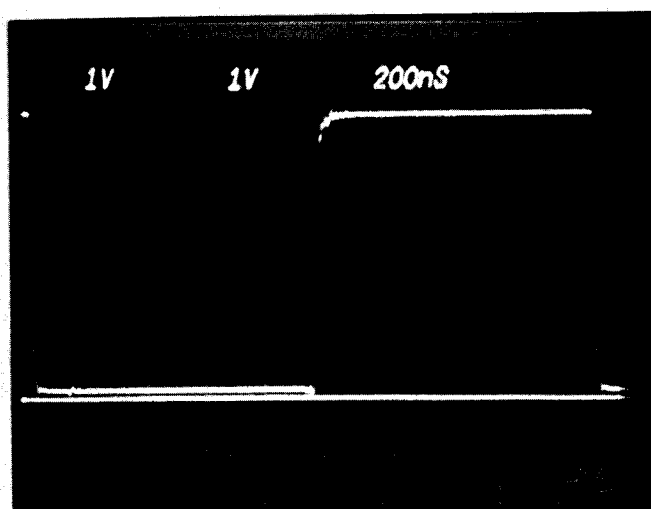
D12-P7



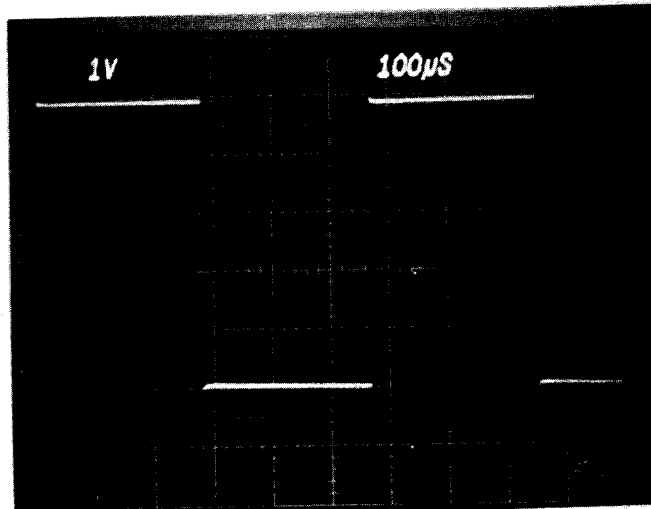
D12-P9



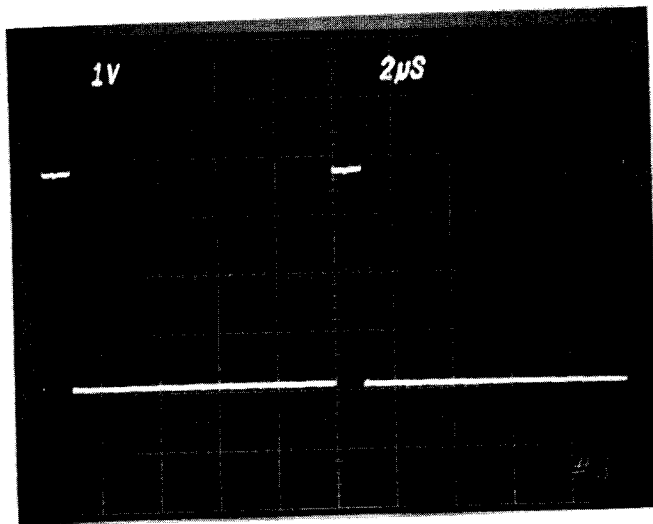
R24&R25 Ø2



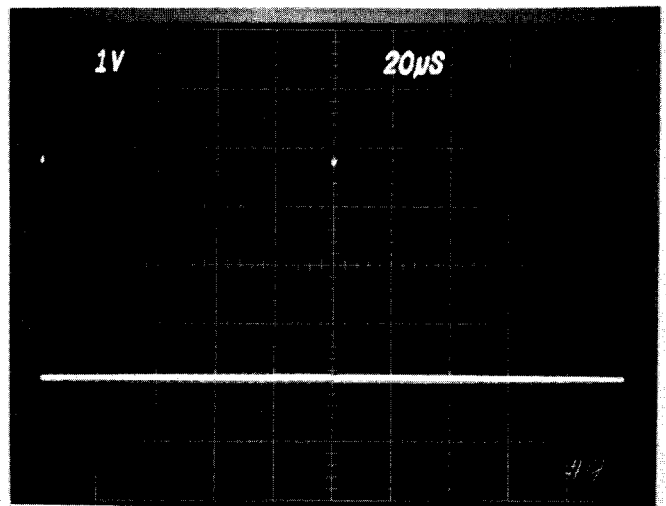
R18&R19 Ø1



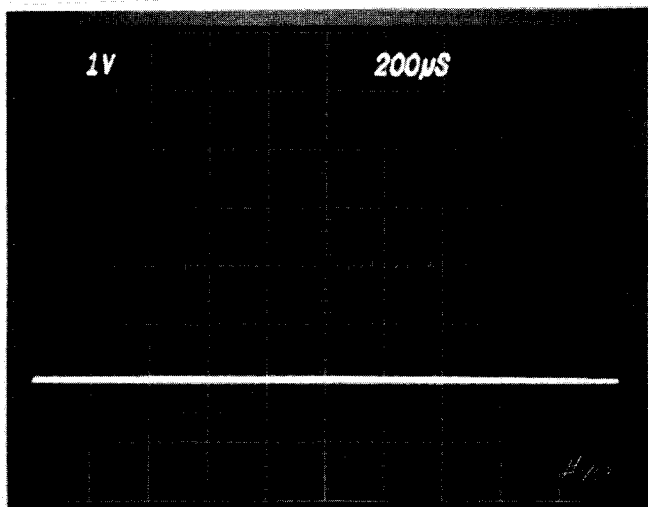
D27-P10



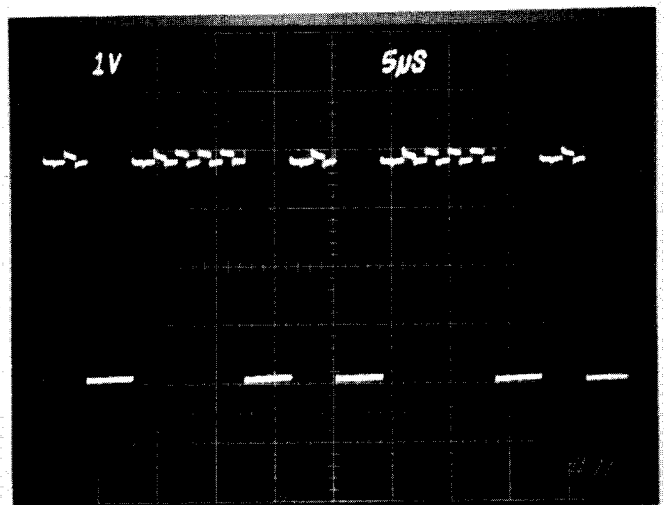
D50-P15



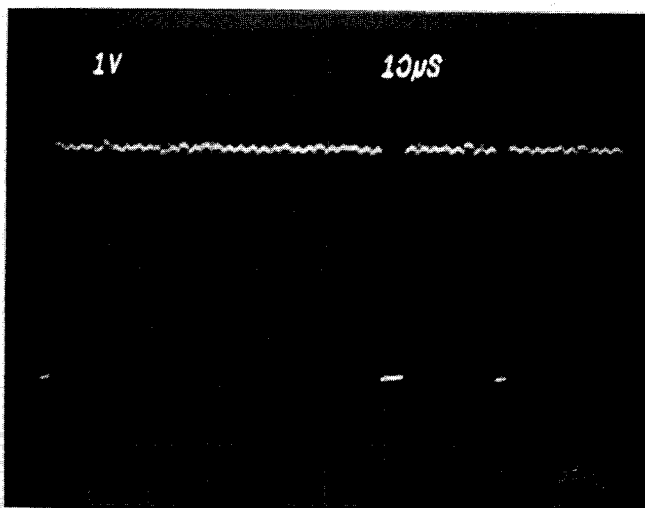
D51-P15



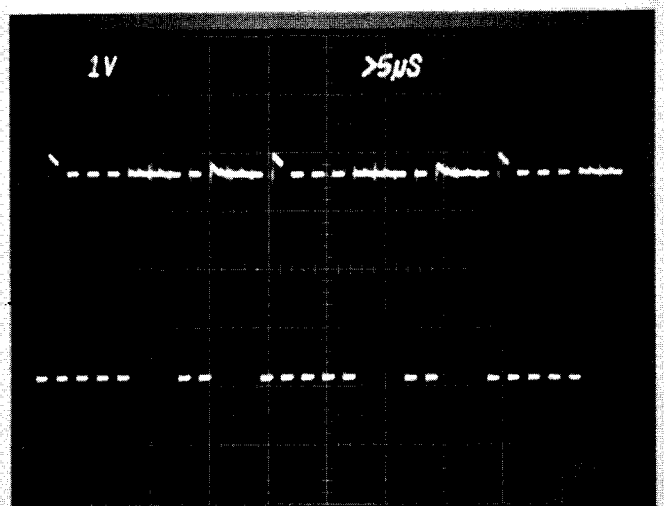
D52-P15



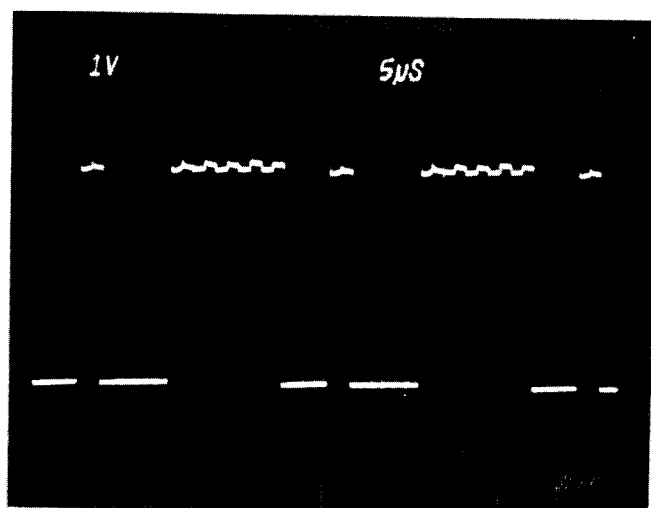
D60-P7 VMA



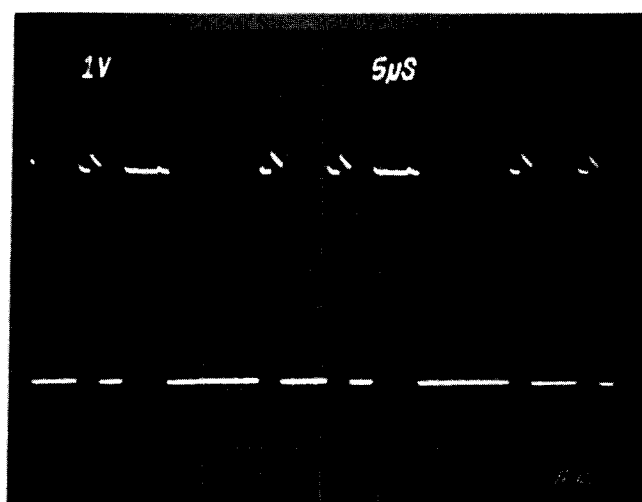
D60-P3 R/W



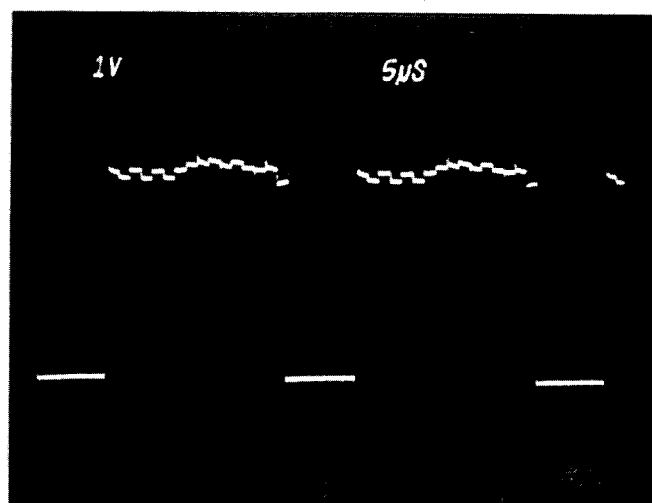
D62-P11



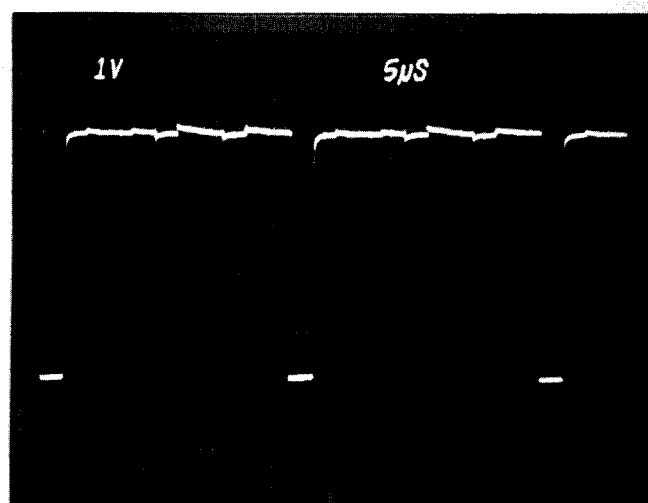
D15-P9



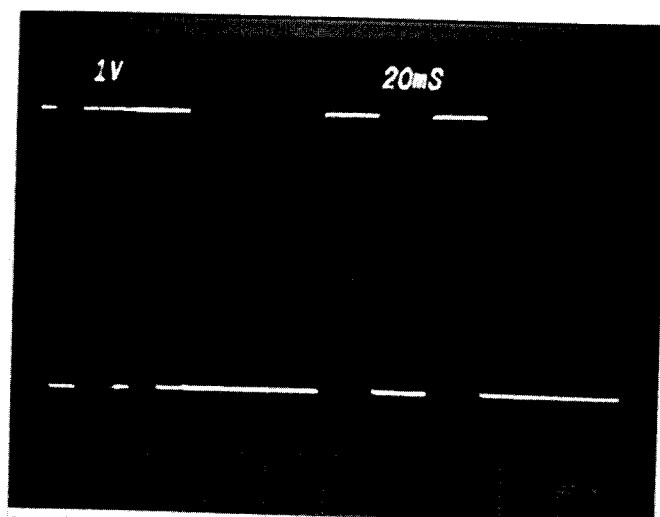
D15-P3



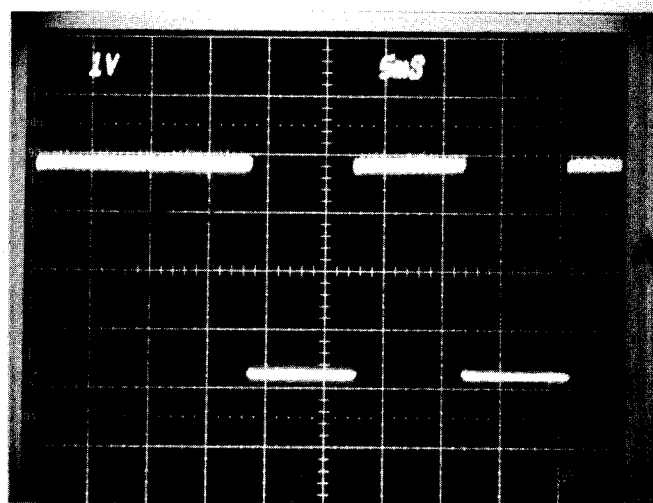
D10-P10



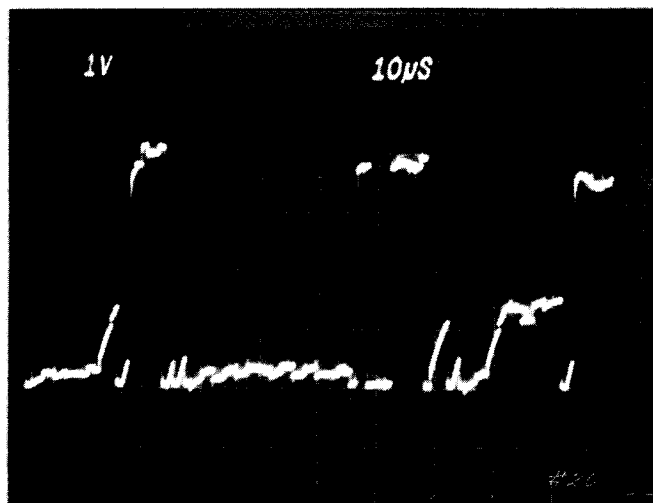
D45-P3



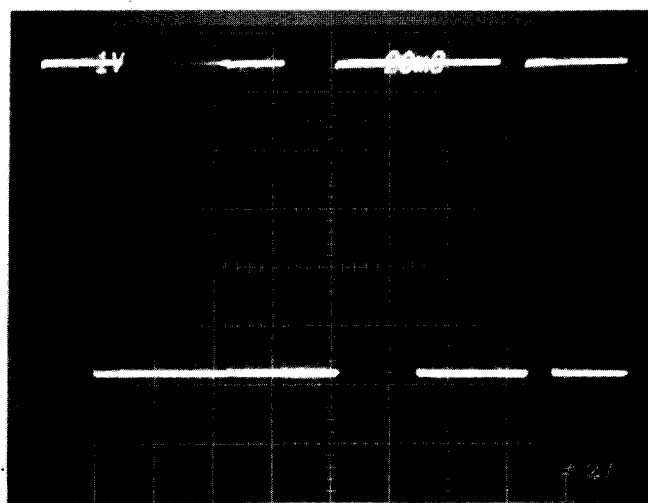
D32-P6



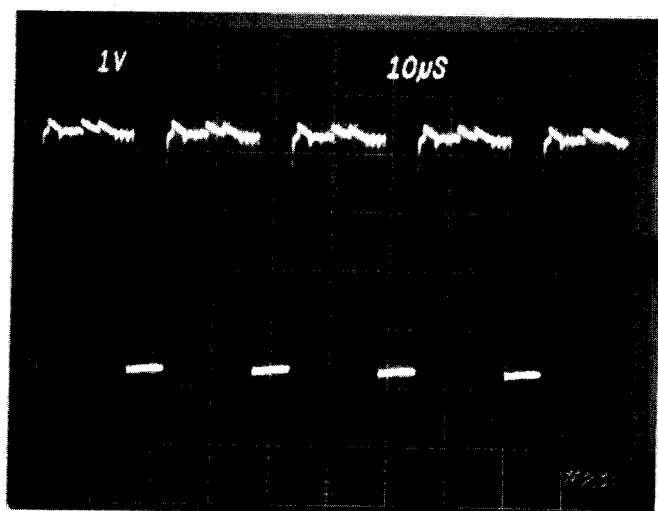
D32-P2



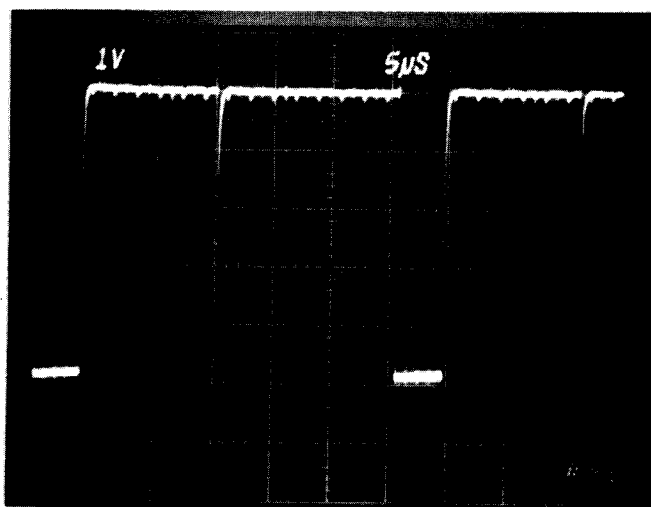
D37-P22



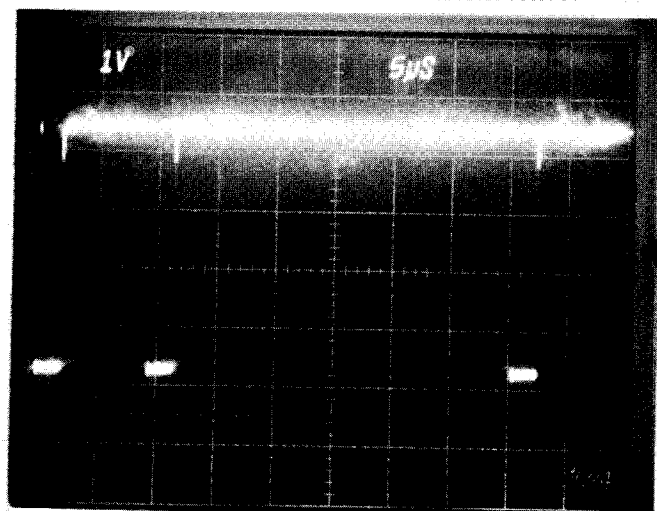
D66-P3



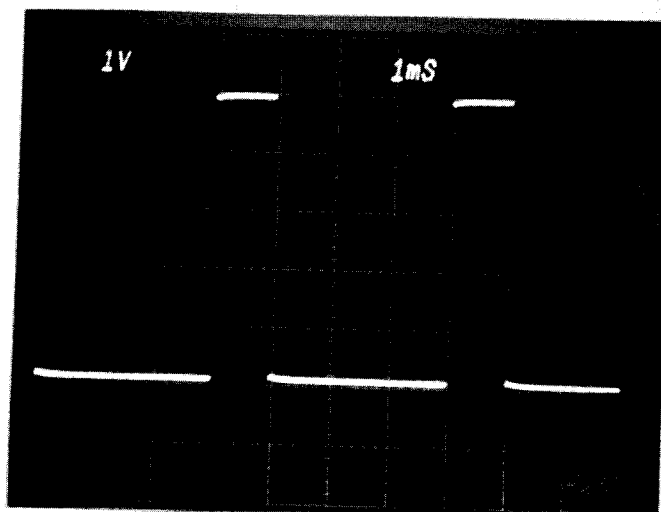
D54-P9



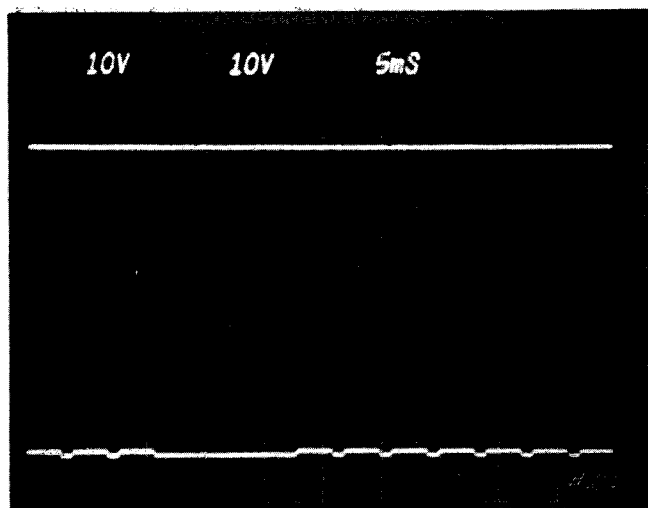
D65-P2



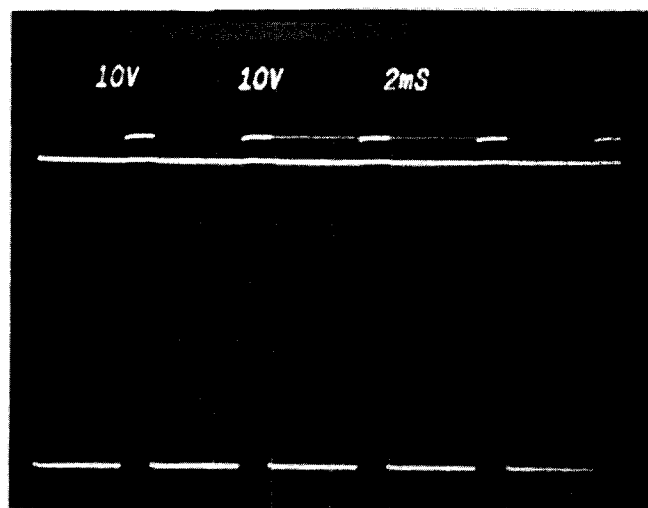
D45-P14



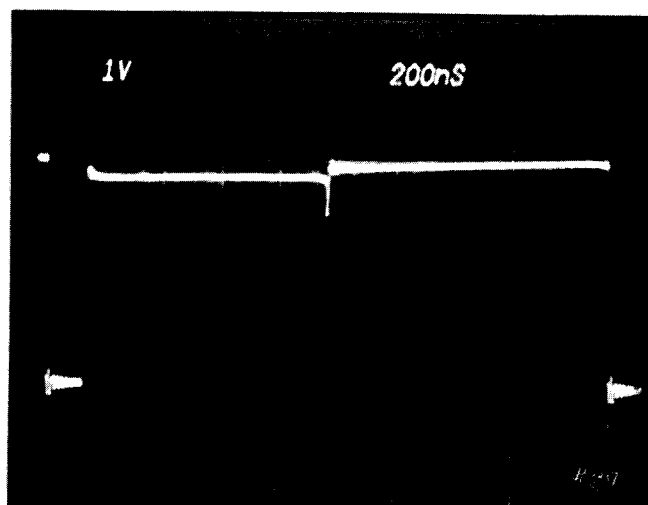
D49-P39



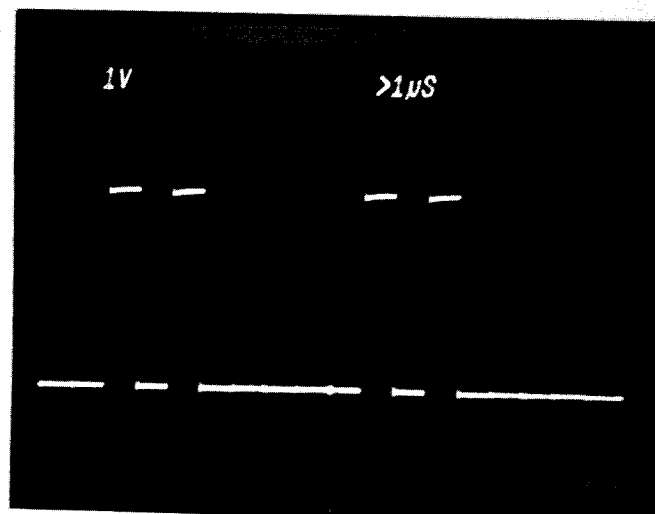
R52&R53



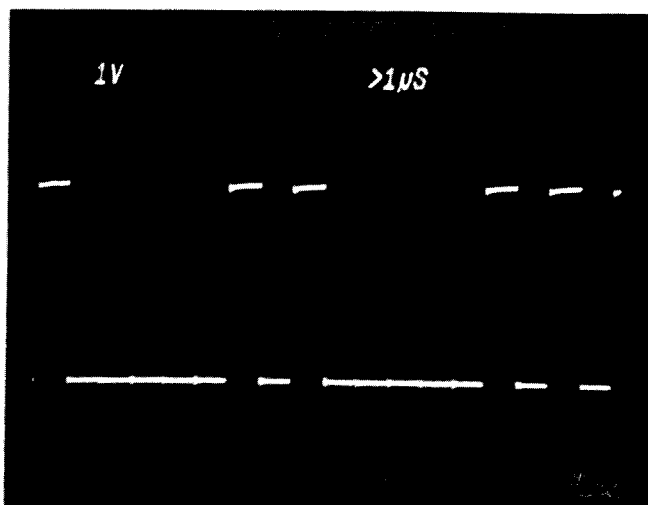
D46-P11



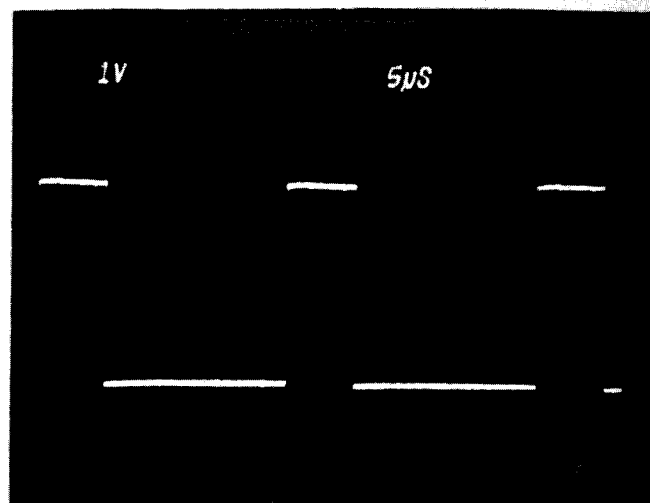
D13-P9



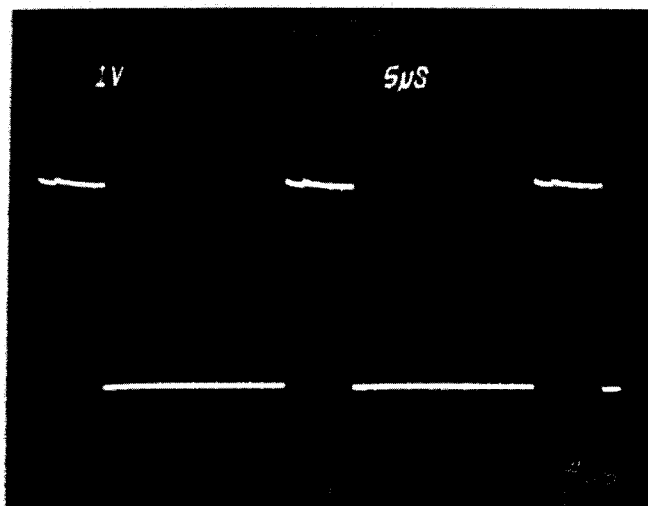
D61-P4



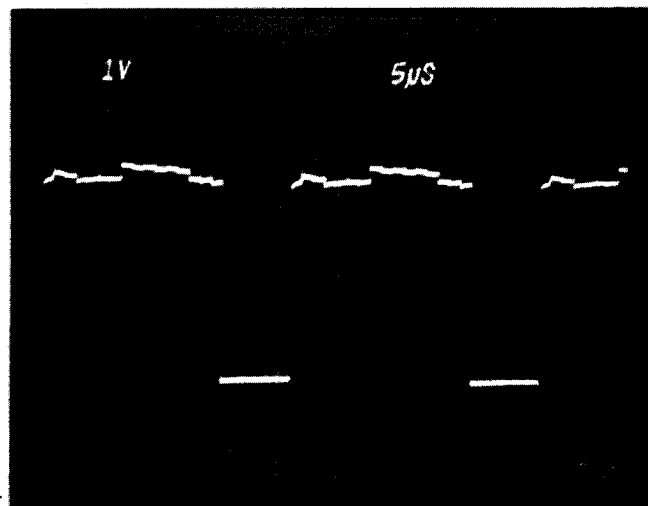
D61-P13



D55-P6



D57-P8



D56-P3

Appendix A

6800 Hex Tape Format

The AMI 6800 Hex Tape format provides a compact representation of binary data patterns for transmission using ASCII communication terminals.

The Hex tape is organized into data records with each record containing information in the same format. The record information consists of type, length, address, data and checksum. All records begin with an 'S' character for start of record identification. All information on the tape which is not between a start of record and the checksum is ignored.

divitive summation (without carry) of the data bytes, the address, and the byte count.

Example Data Record

Memory Contents

Address	Data
A000	10
A001	1A
A002	20
A003	2A

TAPE FORMAT

ASCII Character	Description
1	Start of record (S)
2	Type of record 0 — Header record 1 — Data record 9 — End of file record
3—4	Byte Count Since each data byte is represented as two hex characters, the byte count must be multiplied by two to get the number of characters to the end of the record. (This includes checksum and address data.)
5, 6, 7, 8	Address Value The memory location where this record is to be stored.
9, ..., N	Data Each data byte is represented by two hex characters.
N+1, N+2	Checksum The one's complement of the ad-

Data Record Contents

Character		Tape	
1	Start of record	53	S
2	Type of record	31	1
3	Byte count	30	0
4		37	7
5		41	A
6		30	0
7	Address	30	0
8		30	0
9	Data byte 1	31	1
10		30	0
11	Data byte 2	31	1
12		41	A
13	Data byte 3	32	2
14		30	0
15	Data byte 4	32	2
16		41	A
17	Checksum	38	8
18		34	4

Checksum
Byte Count * 2



The format for all hex tape records is diagrammed below.

Character		Header Record		Data Record		End-of-File Record	
1	Start of Record	53	S	53	S	53	S
2	Type of Record	30	0	31	1	39	9
3	Byte Count	31	12	31	16	30	03
4		32		36		33	
5	Address (if any)	30	0000	31	1100	30	0000
6		30		31		30	
7		30		30		30	
8		30		30		30	
9	Data	34		39	98	46	FC
10		38		38		43	
.		34		30	02		(Checksum)
.		34		32			
.		35					
.		32					
.				41			
.				48	A8 (Checksum)		
N	Checksum	39	9E				
		45					

Appendix B

Parts List for Prototyping Board

Qty	Part	Designation	Qty	Part	Designation
1	AMI S6800 Prototyping Board		3	74160 Asynchronous Decimal Counter	IC 50,51,52
1	AMI S6800 Micro-processor	IC 16	2	74S257 Quad 2-bit Multiplexer	IC 31,42
8	AMI S6810-1 RAM	IC 19,20,21, 22,33,34, 35,36	1	1488 RS-232 Driver	IC 67
3	AMI S6820 PIA	IC 47,48,49	1	1489A RS-232 Receiver	IC 68
1	AMI S6830-003 ROM	IC 10	1	4702 Baud Rate Generator (Fairchild)	IC 27
1	AMI S6830-004 ROM	IC 11	1	555 Voltage Controlled Oscillator	IC 38
*1	AMI S6834 EPROM	IC 6,7,8,9,46	2	96S02 Dual One-Shot	IC 12,13
1	AMI S6850 ACIA	IC 37	14	8T97 Hex Tristate Buffer	IC 1,2,3,4,5, 15,17,18, 24,25,26, 30,60,64
2	74S00 Quad 2-input NAND	IC 56,63	2	DIP Switch, 8 position	IC 32,43
2	74S02 Quad 2-input NOR	IC 39,61	1	DIP Switch, 4 position	IC 28
1	7404 Hex Inverter	IC 14	1	Transistor 2N3563	Q5
1	7407 Hex Open Collector Driver	IC 65	1	Transistor 2N4402	Q6
1	74S08 Quad 2-input AND	IC 62	1	Transistor 2N5400	Q7
2	74S10 Triple 3-input NAND	IC 55,59	2	Transistor 2N5771	Q2,Q4
1	74S20 Dual 4-input NAND	IC 53	2	Transistor 2N5772	Q1,Q3
1	74S30 8-input NAND	IC 40	1	Transistor MJE340 (Motorola)	Q8
2	74LS30 8-input NAND	IC 29,41	2	Diode 1N914	CR 1,2
2	74S32 Quad 2-input OR	IC 57,58	7	Diode 1N4003	CR 3,4,5,6, 7,8,9
1	7437 Quad 2-input NAND Buffer	IC 23	3	Diode A15F (G.E.)	CR 10,11,12
1	7438 Quad 2-input NAND Open Collector	IC 66	1	Crystal 2.4576 MHz	XTAL
3	74S138 3 to 8 bit Decoder	IC 44,45,54	1	Crystal 1.000 MHz	XTAL
			1	Switch, Momentary, C&K 8125A Rt. Angle	Reset
			2	Resistor Packs, 8 resistors, 16 pin, 4.7K	RP1, RP2
			2	Potentiometers, 15 turn, 20K (Bourns 3006P)	R1, R2
			1	Resistor, 10Ω ¼W ± 5% carbon film	R 55

*Sockets required for 5 S6834 EPROMs.

PARTS LIST FOR PROTOTYPING BOARD (CONT'D)

Qty	Part	Designation	Qty	Part	Designation
1	Resistor, 22Ω ¼W ± 5% carbon film	R 24	1	Capacitor, Ceramic Disc .047μF, 6V or more	C 36
1	Resistor, 33Ω ¼W ± 5% carbon film	R 18	31	Capacitor, Ceramic Disc .1μF, 6V or more	C 7,8,9,12, 16,21,23, 24,25,26, 27,28,29, 32,33,34, 35,37,38, 39,40,42, 43,44,45, 46,47,48, 50,51,52,54
2	Resistor, 51Ω ¼W ± 5% carbon film	R 19,25			
1	Resistor, 100Ω ¼W ± 5% carbon film	R 49			
1	Resistor, 120Ω ¼W ± 5% carbon film	R 37			
1	Resistor, 150Ω ¼W ± 5% carbon film	R 45			
4	Resistor, 470Ω ¼W ± 5% carbon film	R 9,10,14, 20	5	Capacitor, Ceramic Disc .1μF, 12V or more	C 10,11,13, 57,58
1	Resistor, 510Ω ¼W ± 5% carbon film	R 44	7	Capacitor, Tantalum, .22μF, 15V	C 22,28,41,49, 55,56,59
10	Resistor, 1KΩ ¼W ± 5% carbon film	R 8,17,23,30, 40,47,48, 50,51,53	1	Capacitor, Tantalum, 4.7μF, 75V	C 53
5	Resistor, 1.5K, ¼W ± 5% carbon film	R 4,6,7,12, 46	1	Socket, 8 pin (TI C930802)	IC 38
2	Resistor, 2K, ¼W ± 5% carbon film	R 41,43	19	Socket, 14 pin (TI C931402)	IC 14,23,29, 39,40,41, 53,55,56, 57,58,59, 61,62,63, 65,66,67, 68
4	Resistor, 3.3K, ¼W ± 5% carbon film	R 13,29,36 42			
15	Resistor, 4.7K, ¼W ± 5% carbon film	R 11,15,16, 21,22,31, 32,33,34, 35,38,39, 40,56,57	25	Socket, 16 pin (TI C931602)	IC 1,2,3,4,5, 12,13,15, 17,18,24, 25,26,27, 30,31,42, 44,45,50, 51,52,54, 60,64
2	Resistor, 10K, ¼W ± 5% carbon film	R 28,52			
1	Resistor, 22K, ¼W ± 5% carbon film	R 54			
2	Resistor, 100K, ¼W ± 5% carbon film	R 3,5	15	Socket, 24 pin	IC 6,7,8,9,10, 21,22,33, 34,35,36, 37
2	Resistor, 1M, ¼W ± 5% carbon film	R 26,27			
4	Capacitor, MICA, 8pF	C 17,18, 19,20	1	Socket, 24 pin (R-N TS-61024)	IC 46
1	Capacitor, MICA, 20pF	C 6			
2	Capacitor, MICA, 56pF	C 30,31	4	Socket, 40 pin (TI C934002)	IC 16,47,48, 49
2	Capacitor, MICA, 100pF	C 2,15			
2	Capacitor, MICA, 270pF	C 1,14	2	Connector # 261-10043-2)	Edge A, B
2	Capacitor, MICA, 470pF	C 3,4			
1	Capacitor, Ceramic Disc .01μF, 6V or more	C 5			

Appendix C

Program Listing

1 PROTO 01/15/76 17:04 PROTO

LOC OBJECT M SOURCE STATEMENT

```

                                TITLE PROTO
                                OPT   LSKP
*****
*
* PROTOTYPE BOARD MONITOR PROGRAM
*
* VERSION 2.0 01/08/76
*
* COPYRIGHT 1976 BY AMERICAN MICROSYSTEMS INC.
*
*****
*
* DEFINITIONS
*
FBCE A ACIAC EQU $FBCE ACIA CONTROL REG
FBCF A ACIAD EQU $FBCF ACIA DATA REG
FBCE A ACIAS EQU $FBCE ACIA STATUS REG
0020 A BLANK EQU $20 BLANK CHAR
000D A CR EQU $0D CARRIAGE RET CHAR
001B A ESC EQU $1B ABORT CHAR
0004 A EOT EQU $04 END OF MSG TO BE PRINTED
FFFF A LAST EQU $FFFF HIGHEST ROM ADDRESS
000A A LF EQU $0A LINE FEED
007F A RUBOUT EQU $7F
*
* EXTERNALS FOR RSRSR AND PROM BURNER
*
DEF MONENT,GETRNG,NXTADR,PXISTS,RNGERR,PBADR
DEF PCRLF,OUTCH,PSPACE,SETMEM,ABORT
DEF PROMAD,ADR,ADDL,ADDH,COUNT,MONITR
REF RSRSR,BURN,MOVE,READ,VFY,PINIT
*
*RSRSR ROUTINE DEFINITIONS*
*
000B A SUBXAB EQU 11 SUBTRACT X FROM A,B
000B A ADDABX EQU 8 ADD A,B TO X
0012 A PMSG EQU 18 PRINT MSG
000F A P2HEX EQU 15 PRINT BYTE AS 2 HEX CHARS
0010 A P4HEX EQU 16 PRINT WORD AS 4 HEX CHARS
0015 A CONMB EQU 21 CONVERT HEX TO BINARY
0011 A PUTA EQU 17 OUTPUT TO ACIA
0014 A GETA EQU 20 INPUT FROM TTY
0013 A ALPHUM EQU 19 TEST FOR ALPHANUMERIC
0009 A PRTXD EQU 9 CONV, X TO DEC, & PRINT
*
* SUBR IS A MACRO TO CALL RSRSR ROUTINES
*
SUBR MACRO PARAM
SWI
BYTE PARAM
MEND
*
*****
*

```




2 PROTO 01/15/76 17:04 PROTO

LOC	OBJECT	M	SOURCE	STATEMENT
			* MONITOR RAM	
			*	
FF90			ORG	8FFFE-110 ***CHANGE IF RAM USAGE CHANGES
	FF90 A		EQU	*
	FF8F A		EQU	*-1 BASE ADR USED WITH INDEX OPS
				BOTTOM OF MONITOR STACK
			*	
FF90	0048		RMB	72 LINE OF TTY INPUT
			*	
	FFD8 A		EQU	*
FFD8	0002		RMB	2 ADDRESS IN PROM
FFDA	0002		RMB	2 OFFSET FOR LOADER/PUNCH
FFDC	0002		RMB	2 PARAM. ENTERED BY USER
FFDE	0002		RMB	2
FFE0	0002		RMB	2
FFE2	0001		RMB	1 POINTER TO LAST CHAR SCANNED
FFE3	0001		RMB	1 TAPE RECORD TYPE
FFE4	0001		RMB	1 COUNT FIELD FROM TAPE
FFE5	0002		RMB	2 CALCULATED CKSM
FFE7	0002		RMB	2 TEMP STORAGE FOR S REG
FFE9	0001		RMB	1 TEMP STORAGE FOR X REG
FFEA	0001		RMB	1 1=ECHO TTY, 0=NO ECHO
				TEMP LOC FOR COUNT
			* USER REGISTERS	
FFEB	0001		RMB	1
FFEC	0001		RMB	1
FFED	0001		RMB	1
FFEE	0002		RMB	2
FFF0	0002		RMB	2
FFF2	0002		RMB	2
			*	
FFF4	0002		RMB	2 USER SWI VECTOR (MAY NOT BE IMPLEMENTED)
FFF6	0002		RMB	2 INDIRECT POINTER TO ACIA FOR RSRSR
FFF8	0002		RMB	2 INTERRUPT REQUEST VECTOR
FFFA	0002		RMB	2 SOFTWARE INTERRUPT VECTOR
FFFC	0002		RMB	2 NON-MASKABLE INTERRUPT VECTOR

3 PROTO 01/15/76 17:04 MONITOR

LOC OBJECT M SOURCE STATEMENT

```

*****
*
* *** MONITOR ENTRY VECTOR ***
*
* RESTART INTERRUPT HANDLER
* INTERRUPT BREAK HANDLER
*
*****
0000                                ISEC
0000 0000 I START EQU *          RESET INTERRUPT HANDLER
0000 20 05 BRA START1
0002 7E 0002 I BREAK EQU *       BREAK ON INTERRUPT ROUTINE
0002 7E 0007 I JMP BREAK1
0005 FBCE A ACIAA WORD ACIAC     POINTER TO ACIA
*
0007 0007 I START1 EQU *
0007 36 PSH A                     SAVE A REG IF STACK EXISTS
0008 07 TPA                       SAVE CONDITION CODES
0009 B7 FFEB A STA A CREG
000C 32 PUL A
000D B7 FFED A STA A AREG         SAVE CURRENT VALUE OF REGS
0010 F7 FFEC A STA B BREG
0013 FF FFEE A STX XREG          SAVE X
0016 BF FFF2 A STS SREG          SAVE SP
0019 8E FF8F A LDS #BOS         INIT. SREG TO MON. STPCK
001C CE 0002 I LDX #BREAK       BREAKPOINT ROUTINE
001F FF FFFC A STX NMIVEC       STORE IN INTERRUPT VECTORS
0022 FF FFF8 A STX IRQVEC
0025 CE 00D1 I LDX #SWI30
0028 FF FFF4 A STX USWI
002B CE 00BE I LDX #SWIHAN      SOFTWARE INTERRUPT HANDLER
002E FF FFFA A STX SWIVEC
0031 CE 0005 I LDX #ACIAA       SET UP ACIA PTR
0034 FF FFF6 A STX ACIAI
0037 86 03 LDA A #3            RESET ACIA
0039 B7 FBCE A STA A ACIAS
*
003C 86 01 LDA A #1            SET ACIA CR
003E B7 FBCE A STA A ACIAC
* PRINT CR,LF, & RETURN TO MONITOR
*
0041 0041 I MONENT EQU *
0041 0041 I MONEN1 EQU *
0041 BD 0005 R JSR PINIT
0044 BD 0304 I JSR PCRLF
*
*****
*
* MONITOR ENTRY POINT
*
*****
0047 0047 I MONITR EQU *
0047 8E FF8F A LDS #BOS         INIT MON. STACK
004A BD 0398 I JSR RDROFF       TURN OFF READER

```

4 PROTO 01/15/76 17:04 MONITOR

LOC	OBJECT	M	SOURCE STATEMENT
004D	B6 FBCF	A	LDA A ACIAD DUMP TTY INPUT DATA
0050	86 3E		LDA A #'> PROMPT USER
0052	BD 02DD	I	JSR OUTCH
			* READ TTY LINE (BUFPTR)
			* STORE TTY INPUT IN BUF UNTIL CR IS HIT
			*
0055	CE FF90	A	LDX #BUF INITIALIZE BUFPTR
0058	FF FFE0	A	STX BUFPTR
005B	0D		SEC SET ECHO FLAG
005C	79 FFE9	A	RCL ECHO
			*BEGIN UNTIL LOOP
005F	8C FFD7	A	RT10 CPX #BUF+71 TEST FOR BUF OVERFLOW
0062	26 02		BNE RT20 NU OVERFLOW
0064	20 47		BRA ABORT
0066	BD 0400	I	RT20 JSR WAITTY READ NEXT CHAR
0069	A7 00		RT30 STA A 0.X INSERT CHAR INTO BUF
006B	08		INX INC BUFPTR
			* WHILE CONDITION :
006C	81 0D		RT90 CMP A #CR CARRIAGE RETURN ?
006E	26 EF		BNE RT10 NU, CONTINUE LOOP
			*END OF LOOP
			*
			* DECODE 1 CHAR COMMAND
			* COMPARE CHAR WITH TABLE OF VALID CHARS FOLLOWED BY
			* ADDRESSES OF APPROPRIATE ROUTINES.
			*
0070	BD 0385	I	JSR PXISTS GET 1ST CHAR
0073	08		INX INC BUFPTR
0074	FF FFE0	A	STX BUFPTR
0077	CE 008C	I	LDX #CTABLE START OF TABLE
			* BEGIN LOOP
007A	A1 00		DLOOP CMP A 0.X COMPARE
007C	26 04		BNE DL10
			* FOUND CHAR. GET ADDRESS IMMEDIATELY FOLLOWING CHAR.
007E	EE 01		LDX 1.X
0080	6E 00		JMP 0.X GO TO PROPER ROUTINE
			* NO COMPARE. MOVE TO NEXT CHAR.
0082	08		DL10 INX
0083	08		INX
0084	08		INX
0085	8C 00AD	I	CPX #CTEND END OF TABLE?
0088	26 F0		BNE DLOOP NU, REPEAT
			* END LOOP.
008A	20 21		BRA ABORT NOT IN TABLE.
			*
			* MONEND EQU MONITR
			*
			* CTABLE: TABLE OF VALID 1 CHARACTER COMMANDS.
			* EACH ENTRY CONSISTS OF 3 BYTES. BYTE 1
			* CONTAINS THE ASCII CHAR. BYTES 2,3 CONTAIN THE
			* ADDRESS OF THE APPROPRIATE ROUTINE.
			*
008C	I		CTABLE EQU *

5 PROTO 01/15/76 17:04 MONITOR

LOC OBJECT M SOURCE STATEMENT

```

008C      4C      BYTE      'L
008D      01AD I    WORD      LOAD
008F      47      BYTE      'G
0090      019F I    WORD      GO
0092      50      BYTE      'P
0093      0308 I    WORD      PUNCH
0095      42      BYTE      'B
0096      0001 R    WORD      BURN
0098      4D      BYTE      'H
0099      0002 R    WORD      MOVE
009B      56      BYTE      'V
009C      0004 R    WORD      VFY
009E      49      BYTE      'I
009F      0003 R    WORD      READ
00A1      53      BYTE      'S
00A2      03E3 I    WORD      SM
00A4      44      BYTE      'D
00A5      0136 I    WORD      DM
00A7      52      BYTE      'R
00A8      00EE I    WORD      PREGS
00AA      45      BYTE      'E
00AB      015D I    WORD      EOF
00AD      00AD I    CTEND     EQU      *
*
*****
*
* ABORT
*
*****
*
00AD      00AD I    ABORT     EQU      *
00AD      00AD I    BADINP    EQU      *
00AD      CE 0273 I    LDX      #MQUES      PRINT 7777
*
* PRINT MSG AND RETURN TO MONITOR
*
00B0      00B0 I    MSGHON    EQU      *
00B0      00B0 I    MSGABT    EQU      *
00B0      8E FF8F A    LDS      #BDS      SI=BOTTOM OF STACK
00B3      3F      +    SUBR     PMSG
00B3      3F      +    SWI
00B4      12      +    BYTE     PMSG
00B5      20 8A      BRA      MONEN1
*

```

6 PROTO 01/15/76 17:04 SWI-HANDLER

LOC OBJECT M SOURCE STATEMENT

```

*****
*
* SWI HANDLER:
*   DETERMINE WHETHER SWI IS MONITOR CALL, RSRSR CALL,
*   OR USER SWI (NOT IMPLEMENTED).
*
*****
*
0087 0087 I BREAK1 EQU * BREAKPOINT ENTRY
0087 BD 0005 R JSR PINIT CLEAR FROM BURNER
00BA 86 80 LDA A #128 PRETEND TO BE SWI 128
00BC 20 1A BRA SWI40 SAVE REGS

00BE I SWIHAN EQU *
* FIND INDEX BYTE (BYTE AFTER SWI THAT GOT US HERE)
00BE 30 TSX
00BF EE 05 LOX 5,X X1=RET, ADR.
00C1 A6 00 LDA A 0,X A1=INDEX BYTE
00C3 2B 0C BMI SWI30 BREAKPOINT?
* IF USER HAS ADDITIONAL (RS)**J ADDR OF FIRST+2 MUST BE IN FFF4
00C5 80 18 SUB A #24 RSRSR CALL?
00C7 2A 03 BPL SWI20 NU ==
00C9 7E 0000 R JMP RSRSR

*
* USER SWI
00CC FE FFF4 A SWI20 LDX USWI
00CF 6E 00 JMP 0,X GU DO IT

*
* MONITOR CALL. COPY REGS FROM STACK
*
00D1 30 SWI30 TSX INCREMENT RET, ADDR.
00D2 6C 06 INC 6,X
00D4 26 02 BNE SWI40
00D6 6C 05 INC 5,X
00D8 CE FFE8 A SWI40 LDX #CREG DEST, FOR 1ST REG
* BEGIN LOOP
00D8 33 SWI50 PUL B GET REG
00DC E7 00 STA B 0,X COPY
00DE 08 INX MOVE TO NEXT REG
00DF 8C FFF2 A CPX #CREG+7 END OF LOOP?
00E2 26 F7 BNE SWI50

* END LOOP
*
* S NOW CONTAINS ITS VALUE BEFORE SWI
* WAS EXECUTED. SAVE IT.
00E4 AF 00 STS 0,X

*
* A STILL CONTAINS SWI INDEX. TEST IT
*
00E6 81 81 CMP A #129
00E8 26 04 BNE PREGS NOT 129: BREAK
00EA 8D 07 BSR PR1 129: SNAPSHOT
00EC 20 1E BRA RESTAK AND RETURN TO USER PROGRAM

```


7 PROTO 01/15/76 17:04 SWI-HANDLER

LOC OBJECT M SOURCE STATEMENT

```

*
*****
* PREGS: PRINT USER REGISTERS
*
*****
*
00EE 8D 03 I PREGS EQU *
00F0 7E 0047 I BSR PR1
00F3 CE FFEB A PR1 EQU * SUBROUTINE TO PRINT REGS
                                X POINTS TO 1ST BYTE OF AREA
                                * PRINT 3 1-BYTE REGS
00F6 C6 03 LDA B #3 SET UP COUNT
*
00F8 PR10 SUBR P2HEX
00F8 3F + SWI
00F9 0F + BYTE P2HEX
00FA 8D 0380 I JSR PSPACE
00FD 5A DEC B
00FE 2E F8 BGT PR10
*
0100 C6 03 * PRINT 3 2-BYTE REGS
                                LDA B #3 SET UP COUNT
*
0102 8D 037C I PR20 JSR P4HEXS
0105 5A DEC B
0106 2E FA BGT PR20
*
0108 8D 0304 I JSR PCRLF PRINT CRLF
010B 39 RTS RETURN
*

```

8 PROTO 01/15/76 17:04 VECTRS

LOC OBJECT M SOURCE STATEMENT

```

*****
*
* RESTORE USER STATUS AND RETURN FROM MONITOR
*
*****
*
* RESTORE USER'S STATUS
*
010C BE FFF2 A RESTAK LDS SREG TOP OF USER STACK
010F CE FFF1 A LDX #CREG+6 USER REGS.
*BEGIN LOOP
0112 A6 00 RUS10 LDA A 0,X GET USER REG
0114 36 PSH A PUSH INTO USER STACK
0115 09 DEX MOVE TO NEXT REG
0116 8C FFEA A CPX #CREG-1 LAST REG ?
0119 26 F7 BNE RUS10 NU. CONTINUE LOOP
*END OF LOOP
011B 3B RTI RETURN TO USER PROG

```

9 PROTO 01/15/76 17:04 COMMANDS

LOC OBJECT M SOURCE STATEMENT

```

*****
*
*  COMMANDS AND SUBROUTINES:
*
*****
*
*****
*  CHEKSM(CKSM)
*  VALIDATE CKSM
*
*****
011C      011C I  CHEKSM      EQU      *
011C      B6 FFE4 A          LDA A    CKSM          SAVE CALC. CKSM
011F      36          PSH A
0120      BD 029E I          JSR      NEXT2D         A1= NEXT BYTE FROM TAPE
0123      33          PUL B
0124      53          COM B          B1= CALC. CKSM
0125      11          CBA          B=TAPE CKSM?
0126      26 01          BNE      CS1              NO.
0128      39          RTS

*
0129      30          CS1      TSX          X1=ADR OF CALC. CKSM
012A      09          DEX
012B          SUBR      P2HEX          PRINT CALC. CKSM
012B      3F          +          SWI
012C      0F          +          BYTE      P2HEX
012D      BD 0380 I          JSR      PSPACE
0130      CE 0278 I          LDX      #MCSER          PRINT "CKSM ERR"
0133      7E 00B0 I          JMP      MSGABT

*
*****
*  DM  ADDL,ADDH  COMMAND
*
*****
0136      0136 I  DM          EQU      *
0136      BD 35          BSR      GETRNG          GET ADD RANGE FROM BUF
                                           RETURNS ADDL,ADDH+1
*
* BEGIN OUTER LOOP
0138      CE FFDC A  DM10      LDX      #ADDL
0138      BD 037C I          JSR      P4HEXS          PRINT ADDL, SPACE
* BEGIN INNER LOOP
013E      FE FFDC A  DM20      LDX      ADDL
0141          SUBR      P2HEX          PRINT MEM(X),SPACE,INC X
0141      3F          +          SWI
0142      0F          +          BYTE      P2HEX
0143      BD 0380 I          JSR      PSPACE
0146      FF FFDC A          STX      ADDL
0149      BC FFDE A          CPX      ADDH          IF ADDL=ADDH+1, END OF RANGE
014C      27 0C          BEQ      DM50          EXIT OUTER LOOP
014E      B6 FFDD A          LDA A    ADDL+1          IF LSB'S OF ADDL=0, END OF LINE
0151      84 0F          AND A    #8F
0153      26 E9          BNE      DM20          NOT END OF LINE. CONTINUE
* END OF INNER LOOP

```

10 PROTO 01/15/76 17:04 COMMANDS

```

LOC  OBJECT  M  SOURCE STATEMENT

0155  BD 0304 I      JSR    PCRLF      PRINT CR,LF
0158  20 DE          BRA     DM10      EXIT INNER LOOP
                                * END OF OUTER LOOP
015A  7E 0041 I      DM50    JMP     MONEN1    CR,LF, BACK TO MONITOR
                                *
                                *****
                                *
                                ** PUNCH END OF FILE AND 60 NULLS
                                *
                                *****
015D  CE 028A I      EOF      LDX     #MPEOF    PUNCH EOF RECORD
0160                                SUBR    PMSG
0160  3F              *      SWI
0161      12          *      BYTE    PMSG
                                *
                                * PUNCH 60 NULLS
                                *
0162  C6 3B          NULLS    LDA     B     #59      LOAD COUNTER
                                *      BEGIN LOOP
0164  4F              NULL1    CLR     A          LOAD NULL
0165  BD 02DD I      JSR     DUTCH    PRINT ONE NULL
0168  5A              DEC     B          DECREMENT COUNTER
0169  26 F9          BNE     NULL1    DONE?
                                *      END OF LOOP
016B  20 ED          BRA     DM50    CR,LF, BACK TO MONITOR
                                *****
                                *
                                * GETRANGE (ADDL,ADDH,BUFPTR)
                                *   GET ADDRESS RANGE FROM BUF
                                *   ABORT IF INVALID
                                *   SET ADDH=ADDH+1 TO SIMPLIFY COMPARISONS
                                *   RETURNS ADDL & ADDH+1
                                *   ALTERS ADR,X,A,B
                                *
                                *****
                                016D I      GETRNG    EQU     *
016D  BD 02B8 I      JSR     NXTADR    GET ADDL
0170  FE FFDA A      LDX     ADR
0173  FF FFDC A      STX     ADDL      STORE ADDL
0176  FF FFDE A      STX     ADDH      MAY BE ONLY 1 PARAM
0179  BD 02B8 I      JSR     NXTADR    GET ADDH
017C  27 06          BEQ     GETRG3    ONLY 1 PARAM
                                *
017E  FE FFDA A      GETRG1    LDX     ADR
0181  FF FFDE A      STX     ADDH      SAVE ADDH
                                * THE NEXT 5 INSTR TEST ADDH=ADDL
0184  CE FF90 A      GETRG3    LOX     #BASE    REF W.R.T. BASE OF RAM
0187  A6 4E          LDA     A     ADDH=BASE,X  MSBYTE
0189  E6 4F          LDA     B     ADDH+1=BASE,X
018B  E0 4D          SUB     B     ADDL+1=BASE,X
018D  A2 4C          SBC     A     ADDL=BASE,X
018F  24 06          BCC     GETRG4
0191  CE 0265 I      RNGERR    LDX     #RNGERR  ADDH,GE,ADDL
0194  7E 00B0 I      JMP     MSGABT    RANGE ERR MSG
                                PRINT MSG & ABORT
                                *

```

11 PROTO 01/15/76 17:04 COMMANDS

LOC	OBJECT	M	SOURCE	STATEMENT
0197	FE FFDE A		GETRG4	LDX ADDH INC ADDH
019A	08			INX
019B	FF FFDE A			STX ADDH
019E	39			RTS
* ***** * GO COMMAND *****				
019F	BD 02B8 I		GO	JSR NXTADR GET PARAM
01A2	27 06			BEQ G10 NU PARAM, CONTINUE EXECUTION
01A4	FE FFDA A			LDX ADR ADR=PARAM FROM NXTADR
01A7	FF FFF0 A			STX PREG
01AA	7E 010C I		G10	JMP RESTAK (IN INTERRUPT HANDLER)
* ***** * LOAD COMMAND *****				
01AD	CE 0000 I		LOAD	EQU * INITIALIZE RANGE & OFFSET
01B0	FF FFD8 A			LDX #0 TU 0000-FFFF,0000
01B3	FF FFDC A			STX OFFSET
01B6	09		LOOFST	STX ADDL
01B7	FF FFDE A			STX ADDH
01BA	BD 02B8 I			JSR NXTADR ANY OPERANDS?
01BD	27 1E			BEQ LHF2 NU, USE DEFAULT.
01BF	FE FFDA A			LDX ADR YES.
01C2	FF FFD8 A			STX OFFSET IF ONE, IT'S OFFSET
01C5	BD 02B8 I			JSR NXTADR ANOTHER?
01C8	27 13			BEQ LHF2 NU.
01CA	FE FFD8 A			LDX OFFSET YES, FIRST TWO ARE RANGE
01CD	FF FFDC A			STX ADDL
01D0	CE 0000 A			LDX #0
01D3	FF FFD8 A			STX OFFSET
01D6	BD A6			BSR GETRG1
01D8	FE FFDE A			LDX ADDH
01DB	20 D9			BRA LOOFST GU TRY AGAIN FOR OFFSET
* BEGIN OUTER LOOP				
01DD	BD 03A5 I		LHF2	JSR RDRON TURN ON READER
* SHORT LOOP TO SKIP HDR RECORDS				
01E0	BD 70		ROPRE	BSR FINDS FIND START OF RECORD
* SETS (ECHO):=0 ON ENTRY				
01E2	BD 0400 I			JSR WAITTY RETURNS (A):=TTY I/P
01E5	81 30			CMP A #10 IGNORE HDR RECORDS
01E7	27 F7			BEQ ROPRE
* END SHORT LOOP				

12 PROTO 01/15/76 17:04 COMMANDS

LOC	OBJECT	M	SOURCE	STATEMENT	
01E9	B7 FFE2	A	STA A	RECTYP	SAVE RECORD TYPE
01EC	7F FFE4	A	CLR	CKSM	
01EF	8D 029E	I	JSR	NEXT2D	READ BYTE COUNT FROM TAPE
01F2	4A		DEC A		DEDUCT ADR & CKSM
01F3	4A		DEC A		
01F4	4A		DEC A		
01F5	B7 FFE3	A	STA A	COUNT	SAVE BYTE COUNT
01F8	8D 029E	I	JSR	NEXT2D	READ ADR FIELD FROM TAPE
01FB	B7 FFDA	A	STA A	ADR	1ST BYTE
01FE	8D 029E	I	JSR	NEXT2D	
0201	BB FFD9	A	ADD A	OFFSET+1	
0204	B7 FFD8	A	STA A	ADR+1	2ND BYTE
0207	B6 FFDA	A	LDA A	ADR	CARRY TO FIRST BYTE
020A	B9 FFD8	A	ADC A	OFFSET	
020D	B7 FFDA	A	STA A	ADR	
0210	B6 FFE2	A	LDA A	RECTYP	GET RECORD TYPE (0,1,9)
0213	81 31		LHF3	CMP A #1	DATA RECORD ?
0215	26 14			BNE LHF4	NU
* LOAD DATA RECORD					
* BEGIN UNTIL LOOP					
0217	BD 029E	I	LDR10	JSR	NEXT2D
* READ 2 HEX DIGITS FROM TAPE, RETURNS IN A					
021A	FE FFDA	A	LDX	ADR	
021D	8D 03AF	I	JSR	SETOFF	STORE IN MEM(X), VERIFY
0220	08		INX		
0221	FF FFDA	A	STX	ADR	
0224	7A FFE3	A	DEC	COUNT	DUES COUNT=0?
0227	2E EE		BGT	LDR10	NU, CONTINUE LOOP
* END UNTIL LOOP					
0229	20 04		BRA	LHF9	
022B	81 39		LHF4	CMP A #19	EOF RECORD ?
022D	26 13			BNE BADTAP	ILLEGAL RECORD TYPE
* LHF9					
022F	BD 011C	I	JSR	CHEKSM	CHECK CKSM
0232	B6 FFE2	A	LDA A	RECTYP	GET RECORD TYPE
0235	81 39		CMP A	#19	EOF RECORD ?
0237	26 A4			BNE LHF2	NU, CONTINUE LOOP
* END OF OUTER LOOP					
0239	BD 039B	I	JSR	RDRUFF	
023C	CE 026F	I	LDX	#MEOF	PRINT "EOF"
023F	7E 00B0	I	JMP	MSGMON	AND RETURN TO MONITR LOOP
* BADTAP					
0242	BD 039B	I	JSR	RDRUFF	
* LHF9					
0245	CE 0281	I	LDX	#MTAPER	PRINT "TAPE ERR"
0248			SUBR	PMSG	
0248	3F		SWI		
0249	12		RYTE	PMSG	
* ACCEPT NO COMMANDS UNTIL USER PRESSES ESC					

13 PROTO 01/15/76 17:04 COMMANDS

LOC	OBJECT	M	SOURCE STATEMENT
024A	7C FFE9	A	INC ECHO SET ECHO
024D	BD 0400	I	BT1 JSR WAITTY ESC CAUSES ABORT
0250	20 FB		BRA BT1

* FIND S			
* READ TAPE UNTIL START OF RECORD			

0252	7F FFE9	A	0252 I FINDS EQU * CLR ECHO NU ECHO
0255	BD 0400	I	*BEGIN LOOP FS10 JSR WAITTY READ NEXT TAPE CHAR
0258	81 53		CMP A #'S CHAR = S
025A	26 F9		BNE FS10 NU
025C	39		*END LOOP RTS
* MESSAGES			
025D	4241		MBADR CHAR /BAD ADR/
025F	4420		
0261	4144		
0263	52		
0264	04		MRNGER BYTE 4
0265	5241		CHAR /RANGE ERR/
0267	4E47		
0269	4520		
026B	4552		
026D	52		
026E	04		MEOF BYTE 4
026F	454F		CHAR /EOF/
0271	46		
0272	04		MQUES BYTE 4
0273	3F3F		CHAR /????/
0275	3F3F		
0277	04		MCSER BYTE 4
0278	4348		CHAR /CKSM ERR/
027A	5340		
027C	2045		
027E	5252		
0280	04		MTAPER BYTE 4
0281	5441		CHAR /TAPE ERR/
0283	5045		
0285	2045		
0287	5252		
0289	04		MPEOF BYTE 4
028A	5339		CHAR /S9030000FC/
028C	3033		
028E	3030		
0290	3030		
0292	4643		

14 PROTO 01/15/76 17:04 COMMANDS

```

LOC   OBJECT  M  SOURCE STATEMENT

0294      04          BYTE 4
0295      0D0A      MCRLFS  BYTE CR,LF,0,0,0,0,'S','1,4
0297      0000
0299      0000
029B      5331
029D      04

*
*****
*
* NEXT 2 DIGITS==
* READ NEXT 2 CHAR FROM TTY TAPE AND CONVERT
* TO HEX NUMBER IN A REG. UPDATE CKSM.
* RETURN UPDATED CKSM IN B REG.
*
*****
029E      029E I  NEXT2D  EQU  *
029E      BD 0400 I      JSR  WAITTY      GET CHAR
02A1      16          TAB          SAVE CHAR IN A
02A2      BD 0400 I      JSR  WAITTY

*
* SET UP PARAMS FOR CONVERSION ROUTINE,
* PUSH ASCII CHARS INTO STACK, POINT X AT STACK.
* SET A=TYPE OF CONVERSION AND B=# OF CHARS TO CONVERT.
*
02A5      36          PSH A
02A6      37          PSH B
02A7      30          TSX
02A8      C6 02      LDA B  #2
02AA      3F          SUBR  CONHB      CONVERT FROM ASCII TO BINARY
02AB      15          SWI
02AC      24 94      BYTE  CONHB
                        BCC  BADTAP      IF NON-HEX CHAR, ABORT

*
02AE      17          TBA          UPDATE CKSM
02AF      FB FFE4 A  ADD B  CKSM
02B2      F7 FFE4 A  STA B  CKSM
02B5      31          INS          RESTORE STACK PTR
02B6      31          INS
02B7      39          RTS

*
*****
*
* NEXT ADR(BUFPTR,ADR)
*
* SET ADR=0 OR NEXT NUMBER SIRING STARTING
* AT BUFPTR
* LEAVES BUFPTR AT CR,DELIMITER,OR FIRST
* CHAR BETWEEN G = Z .
* LEAVES (A)= LAST CHAR SCANNED.
* LEAVES (B)= LS BYTE OF ADR
*
* RETURNS: CC= Z FOR NO PARAMETER
*          ABORTS IF NON-HEX PARAMETER
*

```

15 PROTO 01/15/76 17:04 COMMANDS

LOC OBJECT M SOURCE STATEMENT

```

*****
02B8      02B8 I  NXTADR  EQU      *
02B8      7F FFDA A          CLR      ADR          AURI= 0
02B8      7F FFDB A          CLR      ADR+1
02BE      8D 0385 I          JSR      PXISTS        IS THERE A PARAMETER?
02C1      26 01          BNE      NA1              YES
02C3      39          RTS              RETURN W/NO PARAM  CC=Z

*
*
* SET UP PARAMS FOR ASCII TO HEX CONVERSION
*
02C4      C6 47      NA1      LDA B   #71          MAX. CHARS TO SCAN
02C6      SUBR      CONHR
02C6      3F          +      SWI
02C7      15          +      BYTE   CONHR
02C8      FF FFED A      STX      BUFPTR
02C8      B7 FFDA A      STA A   ADR          SAVE RESULT
02CE      F7 FFDB A      STA B   ADR+1
02D1      A6 00          LDA A   0,X          CHECK TERMINATOR
02D3      SUBR      ALPNUM          IS CHAR ALPHA?
02D3      3F          +      SWI
02D4      13          +      BYTE   ALPNUM
02D5      25 01          BCS      NA3          YES
02D7      39          RTS

*
02D8      7E 00AD I  NA3      JMP      ABORT          NU
*
*****
*
* OUTCH = PRINT CHAR IN A
* OUTCHX = PRINT CHAR AT MEM(X)
* IF CHAR = 'CR', FOLLOW WITH LF & 4 NULLS
*
*****
02D8      A6 00      OUTCHX   LDA A   0,X          ENTRY 1
*
02D0 I      OUTCH      EQU      *
* FIRST CHECK FOR ESC
02DD      37          PSH B
02DE      F6 FBCE A      LDA B   ACIAS          ACIA INPUT STATUS
02E1      57          ASR B          CI=RDRF
02E2      24 0A          BCC      0C10          NU INPUT
02E4      F6 FBCF A      LDA B   ACIAD          READ ACIA
02E7      C1 18          CMP B   #ESC
02E9      26 03          BNE      0C10          NOT ESC
02EB      7E 00AD I      JMP      ABORT

*
02EE      0C10      SUBR      PUTA          PRINT CHAR
02EE      3F          +      SWI
02EF      11          +      BYTE   PUTA
02F0      81 0D          CMP A   #CR
02F2      26 0E          BNE      0C20          NOT CR. RETURN
*

```

16 PROTO 01/15/76 17:04 COMMANDS

LOC	OBJECT	M	SOURCE STATEMENT
02F4	86 0A		LDA A #LF PRINT LF
02F6			SUBR PUTA
02F6	3F	+	SWI
02F7	11	+	BYTE PUTA
02F8	4F		CLR A PRINT 4 NULLS
02F9	C6 04		LDA B #4
			* BEGIN LOOP
02FB			OCLOOP SUBR PUTA
02FB	3F	+	SWI
02FC	11	+	BYTE PUTA
02FD	5A		DEC B
02FE	26 FB		BNE OCLOOP
			* END LOOP
0300	86 0D		LDA A #CR RSTORE A
			*
0302	33		OC20 PUL B
0303	39		RTS
			*

			* PRINT CR,LF,NULL
			*

0304	86 0D		PCRLF LDA A #CR
0306	20 05		BRA OUTCH OUTCH PRINTS LF AFTER CR
			*

			* PUNCH ADDL,ADDH
			* PUNCH MEMORY CONTENTS BETWEEN ADDL & ADDH
			* IN HEX FORMAT
			*

0308	8D 016D I		PUNCH JSR GETRNG READ ADDL & ADDH+1
0308	CE 0000 A		LDX #0
030E	FF FF08 A		STX OFFSET
0311	8D A5		BSR NXTADR ANY OFFSET?
0313	27 06		BEQ PHF15 NU.
0315	FE FF0A A		LDX ADR YES.
0318	FF FF08 A		STX OFFSET
			*
			* PUNCH DATA RECORDS UNTIL ADDL = ADDH
			*
	0318 I		PHF15 EQU *
			* BEGIN LOOP
			*
			* CALCULATE DATA LENGTH = MIN(30, ADDH+1-ADR)
			*
0318	F6 FFDF A		PHF20 LDA B ADDH+1 B:=ADDH-ADDL
031E	F0 FFDD A		SUB B ADDL+1
0321	86 FFDE A		LDA A ADDH
0324	B2 FFDC A		SBC A ADDL
0327	26 04		RNE PUND10 DIFF .GT. 256
0329	C1 1E		CMP B #30 LS BYTE .GT. 30?

17 PROTO 01/15/76 17:04 COMMANDS

LOC	OBJECT	M	SOURCE	STATEMENT	
032B	23 02		BLS	PUND20	
032D	C6 1E		PUND10	LDA B #30	DIFF .GT.30
032F	5C		PUND20	INC B	COUNT1=COUNT+3
0330	5C			INC B	...INCLUDES ADDR & CKSM
0331	5C			INC B	
0332	F7 FFE3 A		STA B	COUNT	
0335	CE 0295 I		LDX	#MCRLFS	
0338			SUBR	PHSG	
0338	3F	+	SWI		
0339	12	+	BYTE	PHSG	
033A	5F		CLR B		B HOLDS CKSM
033B	CE FFE3 A		LDX	#COUNT	PUNCH COUNT
033E	8D 34		BSR	PUNBYT	
0340	37		PSH B		
0341	FE FFDC A		LDX	ADDL	COMPUTE OFFSET ADDRESS
0344	86 FFD8 A		LDA A	OFFSET	
0347	F6 FFD9 A		LDA B	OFFSET+1	
034A			SURR	ADDABX	
034A	3F	+	SWI		
034B	08	+	BYTE	ADDABX	
034C	FF FFDA A		STX	ADR	PUNCH FROM ADR
034F	CE FFDA A		LDX	#ADR	
0352	33		PUL B		
0353	8D 1F		BSR	PUNBYT	(INCREMENTS X)
0355	8D 1D		BSR	PUNBYT	
0357	FE FFDC A		LDX	ADDL	RESTORE X
			* PUNCH BYTES FROM MEMORY UNTIL COUNT IS EXHAUSTED		
			* BEGIN LOOP		
035A	8D 18		PREC10	BSR	PUNBYT (CC=0 IF COUNT=0)
035C	2E FC		BGT	PREC10	
			* END LOOP		
035E	FF FFDC A		STX	ADDL	SAVE X
0361	CE FFE4 A		LDX	#CKSM	PUNCH CKSM
0364	53		COM B		
0365	E7 00		STA B	0,X	CKSM1=B
0367	8D 0B		BSR	PUNBYT	
0369	FE FFDC A		LDX	ADDL	
036C	8C FFDE A		CPX	ADDH	
036F	26 AA		BNE	PHF20	
			* END LOOP		
0371	7E 0041 I		JMP	NONEN1	

			* PUNBYT (MEM(X),COUNT,CKSM)		
			* PUNCH BYTE AT MEM(X) AND ADJUST COUNT AND CKSM.		
			* CC=Z IF COUNT=0		

0374	EB 00		PUNBYT	ADD B 0,X	CKSM1=CKSM+MEM(X)

18 PROTO 01/15/76 17:04 COMMANDS

```

LOC  OBJECT  M  SOURCE STATEMENT

0376                                SUBR  P2HEX          PRINT MEM(X) AS 2 CHAR
0376  3F                                +      SWI
0377  OF                                +      BYTE  P2HEX
0378  7A FFE3 A                        DEC  COUNT
0378  39                                RTS

*****
*
* P4HEXS:  PRINT 2 BYTES AT X AS 4 HEX CHARS + 2 SPACES
*
*****
*
037C  P4HEXS  SUBR  P4HEX
037C  3F                                +      SWI
037D  10                                +      BYTE  P4HEX
037E  80 00                                BSR  PSPACE

*****
*
* PSPACE---PRINT 1 BLANK
*
*****
0380  86 20  PSPACE  LDA  A  #BLANK
0382                                SUBR  PUTA
0382  3F                                +      SWI
0383  11                                +      BYTE  PUTA
0384  39                                RTS

*****
*
* PARAM EXISTS(BUFPTR)  (#BUFPTR) = BUFPTR
*                        (X) = BUFPTR
* INC BUFPTR UNTIL CHAR = ALPHA OR CR
* LEAVE A = MEM(BUFPTR)
* SET Z IF NO PARAMETER EXISTS
*
*****
0385  0385 I  PXISTS  EQU  *  ENTRY FOR (#BUFPTR)=BUFPTR
0385  FE FFE0 A  LDX  BUFPTR
0388 I  PXISTX  EQU  *  ENTRY FOR (X) = BUFPTR
*BEGIN LOOP
0388  A6 00  PX1  LDA  A  0,X  IS CHAR ALPHANUM ?
038A                                SUBR  ALPNUM
038A  3F                                +      SWI
038B  13                                +      BYTE  ALPNUM
038C  25 07  RCS  PX2  YES, EXIT LOOP
038E  81 0D  CMP  A  #CR  IS CHAR CR ?
0390  27 03  BEQ  PX2  YES, EXIT LOOP
0392  08  INX
0393  20 F3  BRA  PX1  MOVE TO NEXT CHAR

*END LOOP
0395  FF FFE0 A  PX2  STX  BUFPTR
0398  81 0D  CMP  A  #CR  SET Z IF NO PARAMETER
039A  39  RTS

*****

```


19 PROTO 01/15/76 17:04 COMMANDS

LOC OBJECT M SOURCE STATEMENT

```

*
* RDR OFF
*   TURNS TAPE RDR OFF!
*   ACIA RTS O/P HIGH
*   ACIA CHAR $13 (DC3)
*
*****
0398 86 01 039B I RDROFF EQU *
039D 87 FBCE A RDR90 LDA A $S01 RTS HIGH
03A0 86 13 STA A ACIAC SET ACIA CONT REG
03A2 SUBR PUTA SEND TTY RDR CONT CHAR
03A2 3F + SWI
03A3 11 + RYTE PUTA
03A4 39 RTS
*****
*
* RDR ON
*   TURNS TAPE READER ON
*   ACIA RTS O/P LOW
*   ACIA CHAR $11 (DC1)
*
*****
03A5 86 41 03A5 I RDRON EQU *
03A7 87 FBCE A RDR90 LDA A $S41 RTS LOW
03AA 86 11 STA A ACIAC SET ACIA CONT REG
03AC SUBR PUTA SEND TTY RDR CONT CHAR
03AC 3F + SWI
03AD 11 + RYTE PUTA
03AE 39 RTS
*****
*
* SETMEM(X)
* SETS MEM(X):=A AND VERIFY
*
*****
03AF 36 03AF I SETOFF EQU *
03B0 86 FFDC A PSH A FIRST CHECK RANGE!
03B3 F6 FFDD A LDA A ADDL LOW LIMIT
03B6 SUBR SUBXAB 16-BIT SUBTRACT
03B6 3F + SWI
03B7 0B + RYTE SUBXAB
03B8 22 0A RHI SETOUT TWO LOW
03BA 86 FFDE A LDA A ADDH HIGH LIMIT
03BD F6 FFDF A LDA B ADDH+1
03C0 SUBR SUBXAB
03C0 3F + SWI
03C1 0B + RYTE SUBXAB
03C2 24 07 BCC SETPUL OK
03C4 32 SETOUT PUL A OUTSIDE RANGE LIMITS
03C5 86 FF LDA A $255 TYPE DELETE (RUBOUT)

```

20 PROTO 01/15/76 17:04 COMMANDS

```

LOC  OBJECT  M  SOURCE STATEMENT

03C7              SUBR  PUTA          TU SIGNAL FACT TO USER
03C7 3F          +      SWI
03C8 11          +      BYTE  PUTA
03C9 20 17      BRA  SETM1          OTHERWISE IGNORE STORE REQUEST
03CB 32          SETPUL PUL A
03CC A7 00      03CC I SETMEM EQU  *
03CE A1 00      STA A 0,X
03D0 27 10      CMP A 0,X          VERIFY
                                ENRROR ?
                                * VERIFY ERROR , PRINT ADR
03D2 FF FFDA A  STX  ADR          SET PARAM FOR P4HEX
03D5 CE FFDA A  LDX  #ADR
03D8 8D C1      BSR  RDROFF
03DA 8D A0      BSR  P4HEXS
03DC CE 025D I  PBADR LDX  #MBADR          PRINT 'BAD ADR'
03DF 7E 0080 I  JMP  MSGABT          PRINT MSG & ABORT

03E2 39          SETM1  RTS
*****
* SM  ADR BYTE1,BYTE2,...
*****
03E3 8D 02B8 I  03E3 I SM  EQU  *
03E6 FE FFDA A  JSR  NXTADR          AURI= NEXT PARAM
03E9 FF FFDC A  SMS  LDX  ADR          SAVE ADR IN ADDL
                                STX  ADDL
* BEGIN WHILE LOOP
03EC 8D 02B8 I  SM10 JSR  NXTADR          AURI= NEXT PARAM
03EF 27 0C      BEQ  SM30          END OF LINE, EXIT LOOP,
03F1 FE FFDC A  LDX  ADDL          X:= ADD TO BE SET
03F4 17          TBA          A:=LS BYTE
03F5 8D 05      BSR  SETMEM          MEM(X)=A, VERIFY
03F7 08          INX          MOVE TO NEXT ADD
03F8 FF FFDC A  STX  ADDL
03FB 20 EF      BRA  SM10
* END OF LOOP
03FD 7E 0047 I  SM30  JMP  MONEND
*****
* WAIT FOR TTY(CHAR,ECHO) (#ECHO)=ECHO
* RETURN NEXT TTY CHAR IN A
* IF (#ECHO) NOT 0 , ECHO CHAR
*****
0400 I WAITTY EQU  *
*LOOP UNTIL INPUT .NE. RUBOUT
0400 W10 SUBR  GETA          READ TTY
0400 3F          +      SWI
0401 14          +      RYTE  GETA
0402 81 18      CMP A #ESC          ESCAPE ?
0404 26 03      BNE  W20          NU

```

21 PROTO 01/15/76 17:04 COMMANDS

LOC OBJECT M SOURCE STATEMENT

```

0406 7E 00AD I      JMP      ABORT      YES, ABORT
0409 81 7F          W20      CMP A      #RUBOUT RUBOUT ?
0408 27 F3          BEQ      W10        YES CONTINUE LOOP
                                *END UNTIL LOOP
040D 7D FFE9 A      TST      ECHO
0410 27 03          BEQ      W30        NU ECHO
0412 8D 020D I      JSR      OUTCH     ECHO A
0415 39            W30      RTS
                                END

```

SYMBOL TABLE:

ABORT	00AD	I	ACIAA	0005	I	ACIAC	FBCE	A	ACIAD	FBCF	A
ACIAI	FFF6	A	ACIAS	FBCE	A	ADDABX	0008	A	ADDH	FFDE	A
ADDL	FFDC	A	ADR	FFDA	A	ALPNUM	0013	A	AREG	FFED	A
BADINP	00AD	I*	BADTAP	0242	I	BASE	FF90	A	BLANK	0020	A
BOS	FF8F	A	BREAK	0002	I	BREAK1	0087	I	BREG	FFEC	A
BT1	024D	I	BUF	FF90	A	BUFPTR	FFE0	A	BURN	0001	R
CHEKSM	011C	I	CKSM	FFE4	A	CONHB	0015	A	COUNT	FFE3	A
CR	000D	A	CREG	FFEB	A	CS1	0129	I	CTABLE	008C	I
CTEND	00AD	I	DL10	0082	I	DLOOP	007A	I	DM	0136	I
DM10	0138	I	DM20	013E	I	DM50	015A	I	ECHO	FFE9	A
EOF	015D	I	EOT	0004	A*	ESC	0018	A	FINDS	0252	I
FS10	0255	I	G10	01AA	I	GETA	0014	A	GETRG1	017E	I
GETRG3	0184	I	GETRG4	0197	I	GETRNG	016D	I	GO	019F	I
IRQVEC	FFF8	A	LAST	FFFF	A*	LOR10	0217	I	LF	000A	A
LHF2	01DD	I	LHF3	0213	I*	LHF4	0228	I	LHF9	022F	I
LOAD	01AD	I	LOOFST	0186	I	MBADR	025D	I	MCRLFS	029F	I
MCSE	0278	I	MEOF	026F	I	MONEN1	0041	I	MONEND	0047	I
MONENT	0041	I	MONITR	0047	I	MOVE	0002	R	MPEOF	028A	I
MQUES	0273	I	MRNGER	0265	I	MSGABT	0080	I	MSGMON	0080	I
MTAPER	0281	I	NA1	02C4	I	NA3	02D8	I	NEXT2D	029E	I
NMIVEC	FFFC	A	NULL1	0164	I	NULLS	0162	I*	NXTAUR	02B8	I
OC10	02EE	I	OC20	0302	I	OCLOOP	02FB	I	OFFSET	FFD8	A
OUTCH	02DD	I	OUTCHX	02DB	I*	P2HEX	000F	A	PAHEX	0010	A
PAHEXS	037C	I	PBADR	03DC	I	PCRLF	0304	I	PHF15	031B	I
PHF20	0318	I	PINIT	0005	R	PMSG	0012	A	PR1	00F3	I
PR10	00F8	I	PR20	0102	I	PREC10	035A	I	PREG	FFF0	A
PREGS	00EE	I	PROMAD	FFD8	A	PRTXD	0009	A*	PSPACE	0380	I
PUNBYT	0374	I	PUNCH	0308	I	PUND10	032D	I	PUND20	032F	I
PUTA	0011	A	PX1	0388	I	PX2	0395	I	PXISTS	0385	I
PXISTX	0388	I*	ROPRE	01E0	I	RDRORF	039B	I	RDRON	03A5	I
READ	0003	R	RECTYP	FFE2	A	RESTAK	010C	I	RNGERR	0191	I
ROF90	039D	I*	RON90	03A7	I*	RSRSR	0000	R	RT10	005F	I
RT20	0066	I	RT30	0069	I*	RT90	006C	I*	RUBOUT	007F	A
RUS10	0112	I	SAVESP	FFE5	A*	SAVEX	FFE7	A*	SETM1	03E2	I
SETHM	03CC	I	SETOFF	03AF	I	SETOUT	03C4	I	SETPUL	03CB	I
SM	03E3	I	SM10	03EC	I	SM30	03FD	I	SM5	03E6	I*
SREG	FFF2	A	START	0000	I*	START1	0007	I	SUBXAB	000B	A
SWI20	00CC	I	SWI30	00D1	I	SWI40	00D8	I	SWI50	00DB	I
SWIHAN	00BE	I	SWIVEC	FFFA	A	TCOUNT	FFEA	A*	USW1	FFF4	A
VFY	0004	R	W10	0400	I	W20	0409	I	W30	0415	I
WAITTY	0400	I	XREG	FFEE	A						

22 PROTO 01/15/76 17:04 COMMANDS

LOC OBJECT M SOURCE STATEMENT

CHECKSUM = 075E

LENGTH OF DSECT = 0 (0000)

LENGTH OF ISECT = 1046 (0416)

NO ERRORS, NO WARNINGS, THIS ASSEMBLY

1 PROM 01/15/76 17:14 PROM BURNER ADDITION TU PROTO

LOC OBJECT M SOURCE STATEMENT

```

*****
*
*       TITLE  PROM BURNER ADDITION TO PROTO
*
* PROM BURNER
*
* VERSION 2.0  01/08/76
*
* COPYRIGHT 1976 BY AMERICAN MICROSYSTEMS INC.
*
*****
*
* ASSEMBLY OPTIONS
*
0001 A MOVER      EQU      1           0= MOVE ROUTINE EXCLUDED
000A A DELAY      EQU      10        PUST PROGRAM DELAY, BEFORE VFY (MS)
                                OPT     LSKP,LMAC
0000                                ISEC
0416                                ORG     $416
                                REF     MOMENT,GETRNG,NXTADR,PXISTS,RNGERR,PBADR
                                REF     PCRLF,PSPACE,SETHEN,ABORT,MONITR
                                REF     PROMAD,ADR,ADDL,ADDH,COUNT
                                DEF     BURN,MOVE,READ,VFY,PINIT
*
* PIA LOCATIONS:
*
FBC0 A PIA        EQU      H'FBC0
0001 A V50        EQU      H'FBC1-PIA
0004 A PROM       EQU      H'FBC4-PIA
*
* STANDARD RAM BUFFER (DEFAULT)
*
FC00 A RAM        EQU      H'FC00
*
* CHARACTER TYPING MACRO
*
TYPE      MACRO CHAR
          IF CHAR 0
          LDAA #CHAR
          IEND
          CALL PRINTA
          MEND
*
* RSRSR CALL MACRO
*
CALL      MACRO ITEM
          SWI
          BYTE ITEM
          MEND
*
* RSRSR CALL LOCATIONS
*
0011 A PRINTA     EQU      17
0010 A P4HEX      EQU      16

```

2 PROM 01/15/76 17:14 PROM BURNER ADDITION TO PROTO

LOC OBJECT M SOURCE STATEMENT

```

*
*   INITIALIZE PROM BURNER PIA'S
*
0416 CE FBC0 A PINIT LDX #PIA
0419 86 38 LDAA #B'00111000 TURN OFF 50V
041B AA 01 ORAA V50,X
041D A7 01 STAA V50,X
041F 86 3A LDAA #B'00111010
0421 A7 05 STAA PROM+1,X R/W TO READ
0423 A7 07 STAA PROM+3,X (HOPE NO DOUBLE-DRIVE HERE)
0425 6F 06 CLR PROM+2,X PROM DATA SET TO INPUTS
0427 6F 04 CLR PROM,X
0429 63 04 COM PROM,X SELECT ADDRESS AS OUTPUTS
042B 86 3E LDAA #B'00111110
042D A7 05 STAA PROM+1,X POINT TO ADDRESS OUTPUT REG.
042F 39 RTS

*
*   TYPE A IN BINARY, ENCLOSED BY SPACES
*
0430 37 P8BIN PSHB SAVE B
0431 36 PSHA
0432 8D 0F BSR PSP PRINT LEADING SPACE
0434 32 PULA
0435 C6 08 LDAB #8 8 DIGIT COUNTER
0437 49 18 ROLA
0438 36 PSHA
0439 86 18 LDAA #24 (=1/2 ASCII "0")
043B 49 ROLA
043C TYPE
      0 + IF 0
      + LDAA #
      + IEND
043C + CALL PRINTA
043C 3F + SWI
043D 11 + BYTE PRINTA
043E 32 PULA
043F 5A DECB
0440 26 F5 BNE 18
0442 33 PULB
0443 7E 0007 R PSP JMP PSPACE PRINT ONE MORE SPACE

*
*   RAM/PROM ADDRESS SETUP & VALIDATION
*
0446 CE FC00 A RASV LDX #RAM INTITALIZE POINTERS TO DEFAULT RAM
0449 FF 000D R STX ADDL
044C CE FE00 A LDXX #RAM+512
044F FF 000E R STX ADDH
0452 7F 000E R CLR COUNT SET FULL PROM FLAG
0455 8D 0003 R JSR PXISTS ...IF NO ADDRESS,
0458 27 06 BEQ 1A1
045A 8D 0001 R JSR GETRNG
045D 7C 000F R INC COUNT
0460 FE 000D R 1A1 LDX ADDL DEFAULT PROM ADDRESS
0463 FF 000B R STX PROMAD IS SAME AS START
0466 8D 0002 R JSR NXTADR TRY FOR PROM ADDRESS

```


3 FROM 01/15/76 17:14 FROM BURNER ADDITION TU PROTO

LOC OBJECT M SOURCE STATEMENT

```

0469 27 06          BEQ    1A3          NO,
046B FE 000C R      LDX    ADR          YES,
046E FF 000B R      STX    PROMAD
0471 CE 000B R 1A3  LDX    #PROMAD      VERIFY THAT RANGE <= 512
0474 0D              SEC                (FORCE BORROW)
0475 E6 07          LDAB   7,X          *ADDH+1
0477 E2 05          SBCB   5,X          *ADDL+1
0479 A6 06          LDAA   6,X
047B A2 04          SBCA   4,X
047D 81 02          CMPA   #2          SHOULD BE 1 OR 0
047F 2C 0B          RGE    1A4          TWO BIG,
0481 EB 01          ADDB   1,X          ALSO SHOULD NOT OVERSTEP FROM
0483 A9 00          ADCA   0,X
0485 A8 00          EORA   0,X
0487 84 FE          ANDA   #H'FE
0489 26 01          RNE    1A4          IT DOES.
048B 39              RTS
048C 7E 0004 R 1A4  JMP     RNGERR      ADDRESS RANGE ERROR

```

* TYPE RAM & ROM ADDRESS & DATA

```

048F CE 000D R VERR LDX    #ADDL      TYPE RAM ADDRESS
0492                CALL   P4HEX
0492 3F              SWI
0493 10              BYTE   P4HEX
0494 FE 000D R      LDX    ADDL      NOW THE BYTE THERE
0497 A6 00          LDAA   0,X
0499 8D 95          BSR     P8BIN
049B B6 FBC6 A      LDAA   PROM+2+PIA THEN FROM DATA
049E 8D 90          BSR     P8BIN
04A0 CE 000B R      LDX    #PROMAD    NOW IF ADDRESS (LOW 8)
04A3 A6 01          LDAA   1,X        DOES NOT MATCH RAM ADDRESS,
04A5 A1 05          CMPA   5,X        *ADDL
04A7 27 02          BEQ     IT
04A9                CALL   P4HEX      PRINT FROM ADDRESS
04A9 3F              SWI
04AA 10              BYTE   P4HEX
04AB 8D 0006 R IT   JSR     PCRLF
04AE 86 40          LDAA   #64
04B0 48              ASLA
04B1 39              RTS              EXIT C=0, Z=0, V=1

```

* PROM ADDRESS SETUP & DATA READ

```

04B2 CE 000B R ADDRS LDX    #PROMAD
04B5 A6 01          LDAA   1,X        LOW 8 BITS
04B7 B7 FBC4 A      STAA   PROM+PIA
04BA A6 00          LDAA   0,X        HIGH BIT
04BC CE FBC0 A      LDX    #PIA
04BF 48              ASLA            POSITION IT
04C0 4C              INCA            WITH DATA REGISTER SELECT
04C1 48              ASLA
04C2 48              ASLA
04C3 A8 07          EORA   PROM+3,X  INSERT INTO CONTROL
04C5 84 0C          ANDA   #12

```

4 PROM 01/15/76 17:14 PROM BURNER ADDITION TU PROTO

LOC	OBJECT	M	SOURCE STATEMENT
04C7	A6 07		EORA PROM+3,X
04C9	A7 07		STAA PROM+3,X
04CB	A6 06		LDAA PROM+2,X
04CD	39		RTS
* PROM VERIFY			
04CE	BD 0446 I	VFY	JSR RASV
04D1	BD 2D	IV	BSR VFY1
04D3	24 02		BCC IN
04D5	BD 3A		BSR JVER
04D7	BD 11	IN	BSR INCAD
04D9	20 F6		BRA IV
* PROM READ			
04DB	BD 0446 I	READ	JSR RASV
04DE	BD 02	IR	BSR ADDRS
04E0	FE 000D R		LDX ADDL
04E3	BD 0008 R		JSR SETMEM
04E6	BD 02		BSR INCAD
04E8	20 F4		BRA IR
* INCREMENT RAM/PROM ADDRESS PUINTERS			
04EA	FE 000B R	INCAD	LDX PROMAD
04ED	08		INX
04EE	FF 000B R		STX PROMAD
04F1	FE 000D R	INK	LDX ADDL
04F4	08		INX
04F5	FF 000D R		STX ADDL
04F8	BC 000E R		CPX ADDH
04FB	26 B5		BNE ADDRS
04FD	7E 000A R	EXIT	JMP MONITR
* PROM DATA VERIFY, ONE BYTE			
0500	BD B0	VFY1	BSR ADDRS
0502	FE 000D R		LDX ADDL
0505	A1 00		CMPA 0,X
0507	27 07		BEO IX
0509	43		COMA
050A	AA 00		ORAA 0,X
050C	43		COMA
050D	26 02		BNE JVER
050F	4C		INCA
0510	39	IX	RTS
0511	7E 048F I	JVER	JMP VERR
* PROM BURNER ROUTINE			
0514	BD 0446 I	BURN	JSR RASV
0517	7D 000F R		TST COUNT
051A	26 1B		BNE IB
051C	7F 000B R		CLR PROMAD

GU SETUP ADDRESSES
VERIFY ONE LOCATION
NU ERROR, OR PRINTED,
PHINT FIXABLE ERROR,
INCREMENT ADDRESSES

SET UP POINTERS
READ ONE BYTE

SIORE IN RAM
NEXTI

EXIT TO MONITOR

SET UP & READ A BYTE
CUMPAE TO RAM

OK: C=0, Z=1, V=0
NU, IS IT FIXABLE?
I.E. NO RAM=0, PROM=1?

YES. C=1, Z=0, V=0

NU, TYPE ERROR

SET UP PARAMETERS
IF FULL, UNPARAMETERIZED,
DU BLANK CHECK



5 FROM 01/15/76 17:14 FROM BURNER ADDITION TO PROTO

LOC	OBJECT	M	SOURCE	STATEMENT
051F	8D 91		1C	BSR ADDR5
0521	26 75			BNE NOGOOD
0523	CE 000B R			LOX #PROMAD
0526	6C 01			INC 1,X
0528	26 F5			BNE 1C
052A	6C 00			INC 0,X
052C	A6 00			LDAA 0,X
052E	46			RORA
052F	25 EE			RCS 1C
0531	6F 00			CLR 0,X
0533	20 02			BRA 1B
0535	8D 83		11	BSR INCAD
0537	C6 03		1B	LDAB #3
0539	8D C5			BSR VFY1
053B	29 C0			BVS EXIT
053D	FE 000D R		1L	LOX ADDL
0540	A6 00			LDAA 0,X
0542	CE FBC0 A			LDX #PIA
0545	A7 06			STAA PROM+2,X
0547	A6 05			LDAA PROM+1,X
0549	84 F7			ANDA #B'11110111
054B	A7 05			STAA PROM+1,X
054D	A6 07			LDAA PROM+3,X
054F	84 FB			ANDA #B'11111011
0551	A7 07			STAA PROM+3,X
0553	6F 06			CLR PROM+2,X
0555	63 06			COM PROM+2,X
0557	37			PSHB
0558	C6 14			LDAB #20
055A	8D 3F			BSR MSEC
055C	8D 3D		1P	BSR MSEC
055E	A6 01			LDAA V50,X
0560	84 F7			ANDA #B'11110111
0562	A7 01			STAA V50,X
0564	8D 35			BSR MSEC
0566	8D 33			BSR MSEC
0568	8D 31			BSR MSEC
056A	8A 38			ORAA #B'00111000
056C	A7 01			STAA V50,X
056E	5A			DECB
056F	26 EB			BNE 1P
0571	6F 06			CLR PROM+2,X
0573	A6 05			LDAA PROM+1,X
0575	8A 38			ORAA #B'00111000
0577	A7 05			STAA PROM+1,X
		10		IF DELAY
0579	C6 0A			LDAB #DELAY
057B	8D 1E		1W	BSR MSEC
057D	5A			DECB
057E	26 FB			BNE 1W
				IEND
0580	33			PULB
0581	8D 0500 1			JSR VFY1
0584	29 0F			BVS 1J
0586	27 AD			BEQ 11

AHA -- IT ISN'T
INCREMENT FROM ADDRESS

ADVANCE TO NEXT
SET TRY COUNTER
CHECK THIS LOCATION
CAN'T PROGRAM 1 TO 0
GET DATUM

SET R/W TO W

TURN IT AROUND
(TO OUTPUTS)

CLEAR TIMER
WAIT 1 MS BETWEEN PULSES

SET HIGH VOLTAGE

3 MS PULSE DURATION

TURN OFF HIGH VOLTAGE

20 TIMES.
CONVERT OUTPUTS TO INPUTS
TURN OFF WRITE

OMIT IF NO POST PROGRAM DELAY

DELAY (8) MS

CHECK IT!
BAD BIT SHOWED UP
GOOD

6 PROM 01/15/76 17:14 PROM BURNER ADDITION TO PROTO

LOC OBJECT M SOURCE STATEMENT

```

0588                                TYPE 21          NU, TYPE A NAK
                                IF 21              0
0588 86 15 +                      LDAA #21
                                +
                                + IEND
058A                                + CALL PRINTA
058A 3F +                      SWI
0588 11 +                      BYTE PRINTA
058C CE FBC0 A                      LDX #PIA
058F 5A                      DECB
0590 26 AB                      BNE IL              AND TRY AGAIN
0592 BD 048F I                      JSR VERR
                                0595 I IJ          EQU *
0595 7E 0005 R JBAD              JMP PBADR          GIVE UP
0598 7E 0009 R NOGOOD           JMP ABORT        PRINT "BAD ADDRESS" AND QUIT
                                *
                                * ONE MILLISECOND DELAY
                                *
0598 6D 05 MSEC                TST PROM+1,X      WAIT FOR CA1 TO FLOP
059D 2A FC                      BPL MSEC
059F A1 04                      CMPA PROM,X
05A1 39                      RTS                  CLEAR IT (WITH A DATA READ)

                                *
                                * MEMORY MOVE
                                *
05A2 BD 0001 R MOVE            IF MOVER
05A5 BD 0002 R              JSR GETRNG          GET SOURCE ADDRESS RANGE
05A8 27 E8                      JSR NXTADR       GET DESTINATION STARTING ADDRESS
05AA FE 000D R IM             BEQ JBAD          ERROR IF NONE
05AD A6 00                      LDAA ADDL        GET BYTE
05AF FE 000C R              LDX 0,X
05B2 BD 0008 R              LDX ADR
05B5 08                      JSR SETMEM          STORE IT WITH VERIFY
05B6 FF 000C R              INX                  INCREMENT POINTERS
05B9 BD 04F1 I              STX ADR
05BC 20 EC              JSR INK
                                BRA IM          COMPARE TO END
                                ELSE            MURE
                                MOVE EQU RAM
                                IEND
                                *
                                * END OF MODULE
                                *
                                END

```

SYMBOL TABLE:

ABORT	0009 R	ADDH	000E R	ADDL	000D R	ADDRS	0482 I
ADR	000C R	BURN	0514 I	COUNT	000F R	DELAY	000A A
EXIT	04FD I	GETRNG	0001 R	INCAD	04EA I	INK	04F1 I
JBAD	0595 I	JVER	0511 I	MONENT	0000 R*	MONITR	000A R
MOVE	05A2 I	MOVER	0001 A	MSEC	0598 I	NOGOOD	0598 I
NXTADR	0002 R	P4HEX	0010 A	P8RTN	0430 I	PBADR	0005 R
PCRLF	0006 R	PIA	FBC0 A	PINIT	0416 I	PRINTA	0011 A
PROM	0004 A	PROMAD	0008 R	PSP	0443 I	PSPACE	0007 R



7 FROM 01/15/76 17:14 FROM BURNER ADDITION TO PROTO

LOC OBJECT M SOURCE STATEMENT

PXISTS	0003	R	RAM	FC00	A	RASV	0446	I	READ	04DB	I
RNGERR	0004	R	SETMEM	0008	R	V50	0001	A	VERR	048F	I
VFY	04CE	I	VFY1	0500	I						

CHECKSUM = 93E6

LENGTH OF DSECT = 0 (0000)

LENGTH OF ISECT = 424 (01A8)

NO ERRORS, NO WARNINGS, THIS ASSEMBLY

1 RSRSR 01/15/76 17:17 RSRSR -- REENTRANT SELF RELATIVE SUBROUTINE ROM

LOC OBJECT M SOURCE STATEMENT

```

                                TITLE RSRSR -- REENTRANT SELF RELATIVE SUBROUTINE ROM
                                *****
                                *
                                * (RS)**3 SUBROUTINE ROM FOR USE WITH PROTO
                                *
                                * VERSION 2.0 01/08/76
                                *
                                * COPYRIGHT 1976 BY AMERICAN MICROSYSTEMS INC.
                                *
                                *****
0018 A NITEMS EQU 24          NUMBER OF ROUTINES
                                *
                                * CALLING SEQUENCE LOC
                                * X SWI
                                * X+1 INDEX
                                * X+2 NEXT INSTRUCTION
                                *
                                ISEC
                                ORG 5BE
                                LOCAL
                                DEF RSRSR
                                *
                                * ENTRY IS VIA LOW ORDER ADDRESS OF ROM
                                * <<ADDRESS IS PLACED IN SWI VECTOR ADDRESS
                                *
05BE I RSRSR EQU *
                                *
                                * GET THE INDEX VALUE
                                * DOUBLE IT FOR VECTOR ADDRESS INDEX
                                *
05BE 30 TSX SP INTO X
                                * RESTORE STATE OF INTERRUPT AT TIME OF CALL
                                *
                                *
05BF EE 05 LDX 5,X X HAS INDEX ADDRESS
                                *
05C1 4F CLR A
05C2 E6 00 LDA B 0,X INDEX INTO B
05C4 58 ASL B DOUBLE
05C5 49 ROL A
                                *
                                * A,B HAS TWO TIMES INDEX
                                *
                                * VECTOR OF SUBROUTINE ADDRESSES IS AT 512 + ROM BASE
                                *
                                * FROM HERE TO VECTOR IS 512 - WHERE WE ARE
05C6 80 00 BSR LOCVV
                                *
                                *
                                *
05C8 I LOCVV EQU * STACK HAS WHERE WE ARE
05C8 30 TSX
                                * A,B WILL HAVE INDEX *2 + LOCATION(1A)
05C9 EB 01 ADD B 1,X
05CB A9 00 ADC A 0,X
                                * ADD VECTOR OFFSET

```

2 RSRSR 01/15/76 17:17 RSRSR -- REENTRANT SELF RELATIVE SUBROUTINE ROM

LOC	OBJECT	M	SOURCE STATEMENT
05CD	CB 24		ADD B #(LOCV&H'00FF) LOW ORDER EIGHT BITS
05CF	89 01		ADC A #(LOCV/H'100) HIGH ORDER EIGHT BITS
			* A,B NOW HAS ADDRESS OF SUBROUTINE ADDRESS
05D1	A7 00		STA A 0,X
05D3	E7 01		STA B 1,X
05D5	EE 00		LDX 0,X
05D7	EB 01		ADD B 1,X
05D9	A9 00		ADC A 0,X
			* ADD IN OFFSET CONTAINED IN VECTOR TABLE
05DB	30		TSX
05DC	A7 00		STA A 0,X
05DE	E7 01		STA B 1,X
05E0	A6 02		LDA A 2,X
05E2	06		TAP
05E3	EE 00		LDX 0,X
			* STORE OLD STATE INTO CC
05E5	31		INS
05E6	31		INS
			* JUMP TO SUBROUTINE
05E7	AD 00		JSR 0,X
			* NORMAL EXIT FROM SUBROUTINE
			* INCREMENT RETURN ADDRESS
05E9	30		TSX
05EA	6C 06		INC 6,X
05EC	26 02		BNE ++4
05EE	6C 05		INC 5,X
			* EXIT
05F0	3B		RTI

3 RSRSR 01/15/76 17:17 SUBROUTINES

LOC	OBJECT	M	SOURCE STATEMENT
			* STACK ELEMENTS ARE STACK POINTER +2 SINCE JSR
0002	A UC		EQU 2 CC RELATIVE TO SP
0003	A UB		EQU 3 B RELATIVE TO SP
0004	A UA		EQU 4 A RELATIVE TO SP
0005	A UXH		EQU 5 XH RELATIVE TO SP
0006	A UXL		EQU 6 XL RELATIVE TO SP
0007	A URH		EQU 7 RH RELATIVE TO SP
0008	A URL		EQU 8 RL RELATIVE TO SP,
			* PUSH ALL UNTO STACK -- REGISTERS CORRECT ON EXIT
0000	A SRH		EQU 0 SYSTEM RETURN H RELATIVE TO SP
0001	A SRL		EQU 1 SYSTEM RETURN L RELATIVE TO SP

4 RSRSR 01/15/76 17:17 PUSH ALL

LOC OBJECT M SOURCE STATEMENT

```

*
*   PUSH ALL REGISTERS UNTO STACK--REGISTERS UNMODIFIED
*
*   CURRENT STACK SP  +1  +2  +3  +4  +5  +6  +7  +8  +9
*                   SRH  SRL  CC  B   A   XH  XL  URL  URH
*   RESULT STACK BEFORE RETURN TO MAIN EXIT
*SRH SRL CC B A XH XL URL URH CC B A XH XL
*
*   LOCAL
05F1 I PUSHALL EQU *
*
*   MAKE SPACE
*
05F1 34      DES
05F2 34      DES
05F3 34      DES
05F4 34      DES
05F5 34      DES
*
*   MOVE STACK DOWN
*
05F6 C6 09      LDA B #9          NINE BYTES TO MOVE
05F8 30          TSX
05F9 A6 05      IS LDA A 5,X      OFFSET OF 5
05FB A7 00      STA A 0,X
05FD 08          INX
05FE 5A          DEC B
05FF 26 F8      BNE IS
*
*   RECOPY "PUSHED" REGISTERS
*
0601 C6 05      LDA B #5          FIVE BYTES TO MOVE
0603 30          TSX
0604 A6 02      IC LDA A UC,X
0606 A7 09      STA A UC+7,X      OFFSET BY 7
0608 08          INX
0609 5A          DEC B
060A 26 F8      BNE IC
*
*   EXIT TO MAIN
*
060C 39          RTS
*
*   USER STACK IS
*                   CC  B   A   XH  XL
*                   SP

```

5 RSRSR 01/15/76 17:17 POP ALL

LOC OBJECT M SOURCE STATEMENT

```

*
*   POP ALL REGISTERS
*
*   LOCAL
060D 30 060D I POPALL EQU *
*   TSX
*   CURRENT STACK
*   SRH SRL CC B A XH XL URH URL CC B A XH XL
*   RESULT STACK
*   SMH SRL CC B A XH XL URH UR
*
*   RECOPY "PULLED" REGISTERS
*
060E C6 05 LDA B #5 FIVE OF THEM
0610 A6 09 IC LDA A UC+7,X OFFSET OF 7
0612 A7 02 STA A UC,X
0614 08 INX
0615 5A DEC B
0616 26 F8 BNE IC
*
*   SHIFT EVERYTHING OVER
*
0618 C6 09 LDA B #9 NINE BYTES
061A A6 03 IS LDA A URL-5,X
061C A7 08 STA A URL,X OFFSET 5
061E 09 DEX
061F 5A DEC B
0620 26 F8 BNE IS
*
*   FINALLY INCREMENT SP
*
0622 31 INS
0623 31 INS
0624 31 INS
0625 31 INS
0626 31 INS
0627 39 RTS

```

6 RSRSR 01/15/76 17:17 TXAB/TABX

LOC OBJECT M SOURCE STATEMENT

```

*
*   TRANSFER      X   TO   A,B
*
0628 30          TXAB    TSX
0629 A6 05          LDA A  UXH,X      X HIGH
062B E6 06          LDA B  UXL,X      X LOW
062D A7 04          STAB   STA A  UA,X    TU   A
062F E7 03          STA B  UB,X      TU   B
*
0631 39          RTS

```

```

*
*   TRANSFER      A,B TO X
*
0632 30          TABX    TSX
0633 A6 04          LDA A  UA,X      A
0635 A7 05          STA A  UXH,X      TU   X HIGH
*
0637 A6 03          LDA A  UB,X      B
0639 A7 06          STA A  UXL,X      TU   X LOW
*
063B 39          RTS

```

7 RSRSR 01/15/76 17:17 XABX

LOC OBJECT M SOURCE STATEMENT

```

*
*   EXCHANGE      A,B AND X
*
063C 30 063C I XABX    EQU   *
*   CURRENT      STACK SRH SRL C   B   A   XH   XL   URH URL
*   RESULT
*
063D A6 05          LDA A  UXH,X      PICK UP UX
063F 36          PSH A
0640 E6 06          LDA B  UXL,X
0642 8D EF          BSR   TABX+1      THEN GO TRANSFER A,B TO X
0644 32          PUL A
0645 20 E6          BRA   STAB        TU STORE IN A,B

```

8 RSRSR 01/15/76 17:17 PUSX/PULX

LOC OBJECT M SOURCE STATEMENT

```

                                LOCAL
                                *   PUSH   X
                                *
0647 I PUSX      EQU      *
                                *   GET   SPACE IN SPACE
                                *
0647 34          DES
0648 34          DES
0649 30          TSX
                                *   MOVE   STACK   DOWN TWO
064A 86 09      LDA A #9      MOVE TOTAL OF 9 BYTES
                                *
064C E6 02      :A          LDA B 2,X
064E E7 00      STA R 0,X
0650 08          INX
0651 4A          DEC A
0652 26 F8      RNE      :A
                                *
                                *   STACK MOVED == INSERT X
0654 30          TSX
0655 A6 05      LDA A UXH,X
0657 A7 09      STA A UXH+4,X
0659 A6 06      LDA A UXL,X
065B A7 0A      STA A UXL+4,X
                                *
065D 39          RTS
                                *   STACK ON RET
                                *   SRH SRL C B A XH XL URH URL XH XL
                                *   SP
                                *
                                *
                                *   PUL X
                                *
                                LOCAL
065E I PULX      EQU      *
                                *   GET X FROM STACK
                                *
065E 30          TSX
065F A6 09      LDA A UXH+4,X      CURRENT X ON STACK
0661 A7 05      STA A UXH,X        REG X
0663 A6 0A      LDA A UXL+4,X
0665 A7 06      STA A UXL,X
                                *
                                *   NOW MOVE UP TWO
                                *   LDA A #9      BYTE COUNT
0667 86 09      LDA A #9
                                *   EQU
0669 E6 08      :A          LDA R 8,X
066B E7 0A      STA R 10,X
066D 09          DEX
066E 4A          DEC A

```

9 RSRSR 01/15/76 17:17 PUSX/PULX

LOC	OBJECT	M	SOURCE STATEMENT
066F	26 F8		BNE IA
		*	UPDATE SP
0671	31		INS
0672	31		INS
		*	
0673	39		RTS

10 RSRSR 01/15/76 17:17 ADDXAB

LOC OBJECT M SOURCE STATEMENT

```

                                LOCAL
                                *
                                *   ADD X   TO   A,B
                                *
                                *
0674 30 0674 I ADDXAB EQU *
0675 8D C6          TSX
0677 8D 03          BSR XABX+1      EASY WAY! EXCHANGE AB & X
0679 20 C2          RSR ADDABX+1    ADD OTHER WAY
                                THEN EXCHANGE BACK
                                BRA XABX+1

                                *
                                *   ADD A,B TO X
                                *
                                *
067B 30            ADDARX TSX
067C A6 03          LDA A UB,X
067E E6 04          LDA B UA,X

                                *
                                *   CODE SHARED BY ADDAX, INDEX
                                *
                                *
0680 AB 06 0680 I ADDAB EQU *
0682 A7 06          ADD A UXL,X      ADD UXL TO UB
                                STORE INTO UXL
                                STA A UXL,X
                                *
0684 E9 05          ADC B UXH,X      ADD UXH TO UA
0686 07            STAUXH TPA        SAVE STATUS
0687 E7 05          STA B UXH,X      STORE INTO UXH
                                *
0689 6D 06          TST UXL,X        TEST LOW BYTE FOR ZERO
                                *
                                *   CODE SHARED BY ADDABX, MUL8, MUL16
                                *
                                *
068B 27 02 068B I TESTZ EQU *
068D 84 FB          BEQ 1A          YES -- HIGH BYTE STATUS IS TRUE RESULT
068F A7 02          1A STA A UC,X    NO -- Z BIT CLEARED
                                SAVE STATUS
                                *
0691 39            RTS

                                *
                                *   ADD A TO X
                                *
                                *
0692 30 0692 I ADDAX EQU *
0693 A6 04          TSX
                                LDA A UA,X
                                *
0695 C6 00 0695 I ADDZ EQU *
                                LDA R #0

```

11 HSRSR 01/15/76 17:17 ADDXAB

LOC OBJECT M SOURCE STATEMENT

0697 20 E7 * ADD B TO X

0699 30 0699 I ADDBX EQU *
 069A A6 03 TSX
 069C 20 F7 LDA A UB,X
 BRA ADDZ

12 RSRSR 01/15/76 17:17 SUBXAB

LOC OBJECT M SOURCE STATEMENT

```

*
*   SUBTRACT X   FROM   A,B
*
069E 30      SUBXAB   TSX
069F 8D 9C      RSR   XABX+1
06A1 8D 03      RSR   SUBABX+1
06A3 20 98      BRA   XABX+1

```

```

*
*   SUBTRACT A,B FROM   X
*
06A5 30      06A5 I SUBABX EQU *
06A6 E6 05      TSX
06A8 A6 06      LDA B UXH,X
06AA A0 03      LDA A UXL,X
06AC A7 06      *
06AE E2 04      SUB A UB,X
06B0 20 04      STA A UXL,X
06B2 20 04      *
06B4 E2 04      SBC B UA,X
06B6 20 04      BRA STAUXX

```

```

*
*   SUBTRACT A   FROM   X
*
06B2 30      06B2 I SUBAX EQU *
06B3 E6 04      TSX
06B5 A6 06      LDA B UA,X
06B7 10      :SUB LDA A UXL,X
06B8 A7 06      SBA
06BA E6 05      STA A UXL,X      SUB A FROM XL
06BC C2 00      STORE          XL
06BE 20 C6      *
06C0 30      LDA B UXH,X
06C1 E6 03      SBC B #0
06C2 20 C6      BRA STAUXX

```

```

*
*   SUB B   FROM X
*
06C0 30      06C0 I SUBBX EQU *
06C1 E6 03      TSX
06C2 20 C6      LDA B UB,X

```

13 RSRSR 01/15/76 17:17 SUBXAB

```

LOC  OBJECT  M  SOURCE STATEMENT
06C3  20 F0          BRA      SUB
      *
      *
      * INDEX:  X=X + A*B      (SAVE USERA,B)
      *
      *
      * LOCAL
      * EQU
      * BSR      MPYB      A*B := USERA*USERB
06C5  80 11 06C5 I INDEX
      *
      * EXCHANGE A & B TO SHARE CODE W/ ADDABX
      *
06C7  37          PSH B
06C8  16          TAB
06C9  32          PUL A
06CA  30          TSX
06CB  20 B3      BRA      ADDAB
      *
      *
      * MULB:  A,B := A*B
      *
      *
      * 06CD I MULB      EQU      *
06CD  80 09      BSR      MPYB
06CF  30          TSX
06D0  E7 03      STA B  UB,X      SAVE RESULT
06D2  A7 04      STA A  UA,X      SET UP N BIT
06D4  07          TPA
06D5  50          TST B
06D6  20 B3      JMPTZ  BRA      TESTZ      UPDATE USER C & RETURN
      *
      *
      * MUL16--16 BIT MULTIPLY
      *   A,B,X := A,B*X
      *
      *
      *   A,B = PARTIAL PRODUCT
      *   USERX = MULTIPLIER
      *   USERA,USERB = MULTIPLICAND
      *
      *
      * 0          IF      NITEMS-24      (UNIT IF NITEMS < 25)
      *
      * LOCAL
      * MUL16  LDA A #16      PUSH COUNTER INTO STACK
      *        PSH A
      *        TSX
      *        CLR A
      *        CLR B
      *        ROR  UXH+1,X  SHIFT LSB INTO CRY
      *        ROR  UXL+1,X
      *
      *
      * LOOP 16 TIMES:
      *
      * LOOP  BCC  ISHIFT  MULTIPLIER IS EVEN
      *        ADD B UB+1,X  A,B := A,B + USERA,B
      *        AND A UA+1,X
      *
      *
      * ISHIFT  ROR A      SHIFT EVERYTHING RIGHT
      *        ROR B

```

14 HSRSR 01/15/76 17:17 SUBXAB

LOC OBJECT M SOURCE STATEMENT

```

ROR UXH+1,X
ROR UXL+1,X
DEC 0,X      DEC. COUNTER
BNE ILOOP

*
* END LOOP
*
INS RESTORE SP
TSX
STA B UB,X
STA A UA,X

*
* SET USER CC: N=N(MSBYTE)
* Z := AND (Z(MSBYTE),...,Z(LSBYTE));
* V := 0;
* CRY := 0.
* THE LAST ADD RESET CRY. STA SET N=N(MSBYTE) & V=0.
*
TPA
ORA B UXH,X   B:= OR OF 3 LS BYTES
ORA B UXL,X

*
* USER CC HAS CORRECT N,V,&C. CC HAS CORRECT Z FOR LS BYTES.
* GO TO END OF ADDXAB TO UPDATE USERC.
*
BRA JMPTZ
IEND

*
* SUBROUTINE MPYB: AA,B :=USERA+USERB
* A = PARTIAL PRODUCT
* B = MULTIPLIER & LSB'S OF PAR. PROD.
* USERA = MULTIPLICAND
*
LOCAL
MPYB LDA A #8          PUSH COUNTER INTO STACK
PSH A
* STACK = COUNT, R, R, R, R, C, B, A, X, X, R, R
CLR A
TSX
LDA B UB+3,X        B=MULTIPLIER
ROR B

*
* LOOP B TIMES:
*
ILOOP BCC ISHIFT     MULTIPLIER IS EVEN
ADD A UA+3,X

*
IShift ROR A          SHIFT LSB OF A INTO B
ROR R
DEC 0,X             CHECK COUNT
BNE ILOOP

*
* END OF LOOP
*

```

06D8 86 08
06DA 36
06DB 4F
06DC 30
06DD E6 06
06DF 56
06E0 24 02
06E2 AB 07
06E4 46
06E5 56
06E6 6A 00
06E8 26 F6

15 RSRSR 01/15/76 17:17 SUBXAB

LOC OBJECT M SOURCE STATEMENT

06EA 31 INS RESTORE SP
06EB 39 RTS

*

16 RSRSR 01/15/76 17:17 SUBROUTINE ADDRESS VECTOR

LOC OBJECT M SOURCE STATEMENT

*
* RELATIVE ENTRY POINTS TO SUBROUTINES VECTOR
*

	0124	A	LOCV	EQU	**LOCVV	
	06EC	I	SVECTOR	EQU	*	INDEX
06EC	FF05	A		WORD	PUSHALL-*	0
06EE	FF1F	A		WORD	POPALL-*	1
06F0	FF38	A		WORD	TXAB-*	2
06F2	FF40	A		WORD	TABX-*	3
06F4	FF48	A		WORD	XABX-*	4
06F6	FF51	A		WORD	PUSX-*	5
06F8	FF66	A		WORD	PULX-*	6
06FA	FF7A	A		WORD	ADDXAB-*	7
06FC	FF7F	A		WORD	ADDABX-*	8
06FE	FF94	A		WORD	ADDAX-*	9
0700	FF99	A		WORD	ADDBX-*	10
0702	FF9C	A		WORD	SUBXAB-*	11
0704	FFA1	A		WORD	SUBABX-*	12
0706	FFAC	A		WORD	SUBAX-*	13
0708	FFB8	A		WORD	SUBBX-*	14
070A	0017	A		WORD	P2HEX-*	15
070C	0010	A		WORD	P4HEX-*	16
070E	002F	A		WORD	PRINTA-*	17
0710	004F	A		WORD	PMESS-*	18
0712	005D	A		WORD	VALAN-*	19
0714	0086	A		WORD	INPUTA-*	20
0716	0096	A		WORD	CONHB-*	21
0718	FFAD	A		WORD	INDEX-*	22
071A	FFB3	A		WORD	MUL8-*	23
	0			IF	NITEMS-24	
				WORD	MUL16-*	24
					IEND	

17 RSRSR 01/15/76 17:17 P4HEX/P2HEX

LOC OBJECT M SOURCE STATEMENT

```

      * LOCAL
      * PRINT 2/4 HEX CHARS FROM MEM(UX,UX+1)
      * UX IS INCREMENTED UPON OUTPUT I.E. UX = UX+2
      *
071C 30 071C I P4HEX EQU *
071D EE 05          TSX
071F 8D 06          LDX UXH,X      USERS X
                      BSR PHEX      PRINT MEM()
      *
      * PRINT 2 HEX CHARS FROM MEM(UX)
      *
0721 30 0721 I P2HEX EQU *
0722 EE 05          TSX
0724 8D 01          LDX UXH,X      USERS X
0726 39            BSR PHEX      PRINT MEM (X)
                      RTS

```

18 RSRSR 01/15/76 17:17 PHEX

LOC OBJECT M SOURCE STATEMENT

```

      *
      * PRINT 2 HEX CHARS FROM MEM(X)
      *
      * LOCAL
0727 A6 00 0727 I PHEX EQU *
0729 8D 29 LDA A 0,X      GET THE CHAR
072B 36    BSR ASCIIIR    CONVERT THE RIGHT NIBBLE AND RESULT IN A
072C A6 00 PSH A          SAVE IT
072E 8D 20 LDA A 0,X      GET CHAR AGAIN
0730 8D 0E BSR ASCIIIL    CONVERT THE LEFT NIBBLE INTO A
0732 32    BSR PUTAX      PRINT A REG CHAR
0733 8D 10 PUL A          RECOVER SAVED
                      BSR PUTA      ...THEN FALL INTO PINCX

```

```

*****
*
* INCREMENT THE USERS X IN THE STACK
*****

```

```

      * LOCAL
0735 30 0735 I PINCX EQU *
0736 6C 08 TSX
0738 26 02 INC UXL+2,X    SP IS +2 SINCE TWO BSR DOWN IN CALLS
073A 6C 07 BNE IRTS      INC MEMORY X LOW
073C 39    INC UXH+2,X    OVER FLOW MEANS INC HIGH PART
                      IRTS   YES == INC HIGH
                      RTS    EXIT

```

19 HSRSR 01/15/76 17:17 PRINT CHAR

LOC OBJECT M SOURCE STATEMENT

```

*
* PRINT THE CHAR IN USERS A
*
073D I PRINTA EQU *
073D 30 TSX
073E A6 04 LDA A UA,X GET CHAR
LOCAL

*
* PRINT CHAR IN DESIGNATED REG
* ACIA ADDRESS IN X
OPT LMAC

*
PUT MACRO BX
PSH BX SAVE REG
IREADY LDA BX 0,X ACIA STATUS
BIT BX #02 READY ?
BEQ IREADY NOT READY
PUL BX RESTORE CHAR
STA BX 1,X PRINT CHAR
RTS
MEND

*
* PRINT CHAR IN A
*
0740 FE FFF6 A PUTAX LDX H'FFF6 GET INDIRECT ADDRESS OF ACIA
0743 EE 00 LDX 0,X GET ACTUAL ADDRESS OF ACIA INTO X
0745 PUTA PUT A
0745 36 + PSH A SAVE REG
0746 A6 00 + IREADY LDA A 0,X ACIA STATUS
0748 85 02 + BIT A #02 READY ?
074A 27 FA + BEQ IREADY NOT READY
074C 32 + PUL A RESTORE CHAR
074D A7 01 + STA A 1,X PRINT CHAR
074F 39 + RTS

```

20 HSRSR 01/15/76 17:17 HEX/TO/ASCII

LOC OBJECT M SOURCE STATEMENT

LOCAL

```

*
*
* CONVERT A FROM HEX TO ASCII LEFT/RIGHT NIBBLE
* LEFT PART
*
0750 I ASCII EQU *
0750 44 LSR A A HAS CHAR TO BE CONVERTED
0751 44 LSR A
0752 44 LSR A
0753 44 LSR A
* INPOSITION
0754 84 0F ASCII AND A #H'0F CLEAR LEFT PART
0756 88 30 ADD A #H'30
0758 81 39 CMP A #H'39 0 TO 9
075A 23 02 RLS IRTS YES DONE
075C 88 07 ADD A #7 NO THEN A TO F
075E 39 IRTS RTS

```

21 RSRSR 01/15/76 17:17 PRINT MESSAGE

LOC OBJECT M SOURCE STATEMENT

```

*
* PRINT MESSAGE POINTED TO BY X AND TERMINATED BY ETX
*
LOCAL
075F I PMESS EQU *
075F 30 TSX
0760 EE 05 LDY UXH,X GET USERS X
0762 A6 00 LDA A 0,X GET CHAR
0764 81 04 CMP A #ETX IS IT TERMINATOR
0766 27 06 BEQ IRTS DONE
0768 8D 06 BSR PUTX PRINT A
076A 8D C9 BSR PINCX INC USERS X
076C 20 F1 BRA PMESS LOOP TILL DONE
076E 39 IRTS RTS
0004 A ETX EQU H'04

```

22 RSRSR 01/15/76 17:17 VALID ALPHA/NUMERIC

LOC OBJECT M SOURCE STATEMENT

```

*
* X HAS ADDRESS OF CHAR TO TO BE TESTED
* FOR BEING ALPHA NUMERIC
* CARRY SET IF TRUE
076F I VALAN EQU *
LOCAL
076F 30 TSX
0770 EE 05 LDX UXH,X GET CHAR ADDRESS
0772 8D 05 BSR ALPNUM TEST MEM(X)=ALPHANUMERIC

*
* SET USER'S CARRY = CURRENT CARRY (AND OTHER FLAGS)
*
0774 07 SCARRY TPA
*
0775 30 SETUS TSX
0776 A7 02 STA A UC,X
0778 39 RTS

*
* SET CARRY IF MEM(X) IS ALPHANUMERIC
* CLEAR V IF HEX DIGIT
*
0779 I ALPNUM EQU *
0779 A6 00 LDA A 0,X GET THE CHAR
077B 81 41 CMP A #'A
077D 2D 0E RLT INUM TWO SMALL FOR ALPHA ,IS IT NUMERIC
077F 81 5A CMP A #'Z
0781 2E 12 BGT INOTOK
0783 81 C7 CMP A #128+'G SET V IF >F
0785 29 10 RVS IRTS QUIT IF NOT HEX (C=1)
0787 80 07 SUB A #7 CONVERT LETTER TO HEX
0789 84 0F IOK AND A #15 STRIP OVERBITS FROM HEX DIGIT
078B 0D SEC SET C FOR VALID A/N
078C 39 RTS

*
078D 81 30 INUM CMP A #'0 NUMERIC TESTING
078F 2D 04 BLT INOTOK NOT NUMERIC
0791 81 39 CMP A #'9
0793 2F F4 RLE IOK IT IS IN 0-9
0795 0C INOTOK CLC RESET CARRY FOR NOT A/N
0796 0B SEV SET V FOR NOT HEX EITHER
0797 39 IRTS RTS

```


23 RSRSR 01/15/76 17:17 INPUT A

LOC OBJECT M SOURCE STATEMENT

```

0798 20 9B      JPINCX  BRA    PINCX          EXTRA BRA TO REACH PINCX
*****
*
* INPUTA:
*   INPUT ACIA DATA INTO A REG
*   STRIP PARITY
*
*****
                                LOCAL
079A  FE FFF6 A  INPUTA      EQU    *
079D  EE 00      LDX      H'FFF6      GET ACIA INDIRECT ADDRESS
                                LDX      0,X      GET ACIA ADDRESS
*
079F  A6 00      IWAIT      LDA A    0,X      ACIA STATUS
07A1  47          ASR A          CARRYI=RDRF
07A2  24 FB      BCC      IWAIT      NU INPUT, LOOP.
*
07A4  A6 01      LDA A    1,X      ACIA DATA
07A6  84 7F      AND A    #H'7F      STRIP PARITY
07A8  30          TSX          PUT RESULT ONTO STACK
07A9  A7 04      STA A    UA,X
07AB  39          RTS
*

```

24 RSRSR 01/15/76 17:17 HEX TO BINARY

LOC OBJECT M SOURCE STATEMENT

```

*****
*
* CONHB--CONVERT HEX TO BINARY:
*   SCAN UP TO R ASCII CHARACTERS STARTING AT X
*   LOOKING FOR A VALID HEX NUMBER.  RETURN BINARY
*   EQUIVALENT OF NUMBER IN A,B.  IF NUMBER HAS MORE THAN
*   16 BITS, IGNORE MSB'S.
*
* INPUT:  X=ADDRESS OF 1ST CHAN TO BE SCANNED.
*         R=MAX. # OF CHARS TO BE SCANNED.
*
* OUTPUT: A,B=BINARY RESULT
*         CARRY=1 IF VALID NUMBER IS FOUND
*         X POINTS TO LAST CHAN SCANNED
*
*****
LOCAL
07AC I CONHB EQU *
07AC 30      TSX
07AD E6 03   LDA B  UB,X      GET MAX COUNT
07AF 6F 04   CLR  UA,X      CLEAR USER'S A,B REGS
07B1 6F 03   CLR  UB,X

*
* LOOP WHILE NOT ALPHANUMERIC AND COUNT > 1
*
07B3 30      !LOOP1 TSX
07B4 EE 05   LDX  UXH,X      GET CHAR ADDRESS
07B6 8D C1   BSR  ALPNUM     IS MEM(X) ALPHANUMERIC?
07B8 25 09   BCS  !FOUND     YES, STOP SCANNING
07BA 5A      DEC B          DEC COUNT
07BB 2F 04   RLE  !ENDCNT    COUNT EXHAUSTED
07BD 8D D9   RSR  JPINCX     INC USER'S X
07BF 20 F2   BRA  !LOOP1

*
* END LOOP
*
* COUNT EXHAUSTED WITH NO SUCCESS.
* (CARRY WAS RESET BY ALPNUM)
*
07C1 20 B1   !ENDCNT BRA  SCARRY    RESET USER C AND RETURN
*
* WHILE HEX AND COUNT > 0 SHIFT MEM(X) INTO UA,UB
*
* BEGIN OUTER LOOP
07C3 30      !FOUND TSX
07C4 EE 05   LDX  UXH,X      CNVT MEM(X) TO HEX
07C6 8D B1   BSR  ALPNUM     INVALID CHAR
07C8 29 18   BVS  !NOGOOD    SAVE COUNT
07CA 37      PSH B          LOOP COUNT
07CB C6 04   LDA B  #4

*
* SHIFT LEFT UA,UB
*
07CD 30      TSX
07CE 68 04   !SLOOP ASL  UB+1,X    +1 TO COMP, FOR PUSH

```

25 RSRSR 01/15/76 17:17 HEX TO BINARY

LOC OBJECT M SOURCE STATEMENT

```

07D0 69 05          ROL    UA+1,X
07D2 5A             DEC B
07D3 2E F9          BGT     ;SLOOP
*
07D5 AA 04          ORA A   UB+1,X      'UR' IN NEW CHAR
07D7 A7 04          STA A   UB+1,X
07D9 33             PUL R             RETRIEVE COUNT
07DA 8D BC          BSR     JPINCX      INC USER X
07DC 5A             DEC B
07DD 2E E4          BGT     ;FOUND      REPEAT
* END OUTER LOOP
07DF 0D             SEC
07E0 20 92          BRA     SCARRY      VALID NUMBER
*                               SET USER C AND RETURN
*
* NON-HEX CHAR FOUND.  IF CHAR = G-Z, THIS IS NOT A VALID
* HEX NUMBER.  OTHERWISE, CHAR IS A DELIMITER AND
* NUMBER IS VALID.
*
07E2 07             INOGOOD TPA          TUGGLE CARRY BIT
07E3 4C             INC A
07E4 20 8F          RRA     SETUS       SETUP USER STATUS & RETURN
*
END

```

SYMBOL TABLE:

ADDAB 0680 I	ADDABX 0678 I	ADDAX 0692 I	ADDBX 0699 I
ADDXAB 0674 I	ADDZ 0695 I	ALPNUM 0779 I	ASCIIIL 0750 I
ASCIIIR 0754 I	CONHB 07AC I	ETX 0004 A	INDEX 06C5 I
INPUTA 079A I	JMPTZ 06D6 I*	JPINCX 0798 I	LOCV 0124 A
LOCVV 05C8 I	MPY8 06D8 I	MUL8 06CD I	NITEMS 0018 A
P2HEX 0721 I	P4HEX 071C I	PHEX 0727 I	PINCX 0735 I
PMESS 075F I	POPALL 06D0 I	PRINTA 073D I	PULX 065E I
PUSHAL 05F1 I	PUSX 0647 I	PUTA 0745 I	PUTAX 0740 I
RSRSR 058E I	SCARRY 0774 I	SETUS 0775 I	SRH 0000 A*
SRL 0001 A*	STAB 062D I	STAXH 0686 I	SUBABX 06A5 I
SUBAX 0682 I	SUBBX 06C0 I	SUBXAB 069E I	SVECTO 06EC I*
TABX 0632 I	TESTZ 0688 I	TXAB 0628 I	UA 0004 A
UB 0003 A	UC 0002 A	URH 0007 A*	URL 0008 A
UXH 0005 A	UXL 0006 A	VALAN 076F I	XABX 063C I

CHECKSUM = C70E

LENGTH OF DSECT = 0 (0000)
LENGTH OF ISECT = 552 (0228)

NO ERRORS. NO WARNINGS. THIS ASSEMBLY

S6800 INSTRUCTION SET

		Addressing Mode								Condition Reg				
Instruction	Mnemonic	Implied	Immediate	Direct	Extended	Indexed	Relative	Boolean/Arith	5 4 3 2 1 0					
		OP MC PB	OP MC PB	OP MC PB	OP MC PB	OP MC PB	OP MC PB	Operation	H	I	N	Z	V	C
Load accumulator	LDAA		86 2 2	96 3 2	B6 4 3	A6 5 2		M → A	•	•	•	•	•	•
	LDAB		C6 2 2	D6 3 2	F6 4 3	E6 5 2		M → B	•	•	•	•	•	•
Load stack pointer	LDS		8E 3 3	9E 4 2	BE 5 3	AE 6 2		M → SPH, (M + 1) → SPL	•	•	•	•	•	•
Load index register	LDX		CE 3 3	DE 4 2	FE 5 3	EE 6 2		M → XH, (M + 1) → XL	•	•	•	•	•	•
Store accumulator	STAA			97 4 2	B7 5 3	A7 6 2		A → M	•	•	•	•	•	•
	STAB			D7 4 2	F7 5 3	E7 6 2		B → M	•	•	•	•	•	•
Store stack pointer	STS			9F 5 2	BF 6 3	AF 7 2		SPH → M, SPL → (M + 1)	•	•	•	•	•	•
Store index register	STX			DF 5 2	FF 6 3	EF 7 2		XH → M, XL → (M + 1)	•	•	•	•	•	•
Transfer accumulators	TAB	16 2 1						A → B	•	•	•	•	•	•
	TBA	17 2 1						B → A	•	•	•	•	•	•
Transfer Acc. to cond. reg.	TAP	06 2 1						A → CCR	Note 12					
Transfer cond. reg. to Acc.	TPA	07 2 1						CCR → A	•	•	•	•	•	•
Transfer stack ptr to index	TSX	30 4 1						SP + 1 → X	•	•	•	•	•	•
Transfer index to stack ptr	TXS	35 4 1						X - 1 → SP	•	•	•	•	•	•
Pull data	PULA	32 4 1						SP + 1 → SP, Msp → A	•	•	•	•	•	•
	PULB	33 4 1						SP + 1 → SP, Msp → B	•	•	•	•	•	•
Push data	PSHA	36 4 1						A → Msp, SP - 1 → SP	•	•	•	•	•	•
	PSHB	37 4 1						B → Msp, SP - 1 → SP	•	•	•	•	•	•
Add accumulators	ABA	1B 2 1						A + B → A	•	•	•	•	•	•
Add	ADDA		8B 2 2	9B 3 2	BB 4 3	AB 5 2		A + M → A	•	•	•	•	•	•
	ADDB		CB 2 2	DB 3 2	FB 4 3	EB 5 2		B + M → B	•	•	•	•	•	•
Add with carry	ADCA		89 2 2	99 3 2	B9 4 3	A9 5 2		A + M + C → A	•	•	•	•	•	•
	ADCB		C9 2 2	D9 3 2	F9 4 3	E9 5 2		B + M + C → B	•	•	•	•	•	•
Subtract accumulators	SBA	10 2 1						A - B → A	•	•	•	•	•	•
Subtract	SUBA		80 2 2	90 3 2	B0 4 3	A0 5 2		A - M → A	•	•	•	•	•	•
	SUBB		C0 2 2	D0 3 2	F0 4 3	E0 5 2		B - M → B	•	•	•	•	•	•
Subtract with carry	SBCA		82 2 2	92 3 2	B2 4 3	A2 5 2		A - M - C → A	•	•	•	•	•	•
	SBCB		C2 2 2	D2 3 2	F2 4 3	E2 5 2		B - M - C → B	•	•	•	•	•	•
Increment	INCA	4C 2 1						A + 1 → A	•	•	•	•	•	•
	INCB	5C 2 1						B + 1 → B	•	•	•	•	•	•
	INC				7C 6 3	6C 7 2		M + 1 → M	•	•	•	•	•	•
Increment stack pointer	INS	31 4 1						SP + 1 → SP	•	•	•	•	•	•
Increment index reg.	INX	08 4 1						X + 1 → X	•	•	•	•	•	•
Decrement	DECA	4A 2 1						A - 1 → A	•	•	•	•	•	•
	DECB	5A 2 1						B - 1 → B	•	•	•	•	•	•
	DEC				7A 6 3	6A 7 2		M - 1 → M	•	•	•	•	•	•
Decrement stack pointer	DES	34 4 1						SP - 1 → SP	•	•	•	•	•	•
Decrement index register	DEX	09 4 1						X - 1 → X	•	•	•	•	•	•
Complement (1's)	COMA	43 2 1						A → A	•	•	•	•	•	•
	COMB	53 2 1						B → B	•	•	•	•	•	•
	COM				73 6 3	63 7 2		M → M	•	•	•	•	•	•
Complement (2's)	NEGA	40 2 1						00 - A → A	•	•	•	•	•	•
	NEGB	50 2 1						00 - B → B	•	•	•	•	•	•
	NEG				70 6 3	60 7 2		00 - M → M	•	•	•	•	•	•
Decimal adjust accumulator	DAA	19 2 1							•	•	•	•	•	•

OP = Operation Code

MC = Number of MPU Cycles

PB = Number of Program Bytes

		Addressing Mode							Condition Reg
Instruction	Mnemonic	Implied	Immediate	Direct	Extended	Indexed	Relative	Boolean/Arith	5 4 3 2 1 0
		OP MC PB	OP MC PB	OP MC PB	OP MC PB	OP MC PB	OP MC PB	Operation	H I N Z V C
Logical and	AND		84 2 2	94 3 2	B4 4 3	A4 5 2		A ← M → A	• • • • • R •
	ANDB		C4 2 2	D4 3 2	F4 4 3	E4 5 2		B ← M → B	• • • • • R •
	ORA		8A 2 2	9A 3 2	BA 4 3	AA 5 2		A ← M → A	• • • • • R •
	ORAB		CA 2 2	DA 3 2	FA 4 3	EA 5 2		B ← M → B	• • • • • R •
	EOR		88 2 2	98 3 2	B8 4 3	A8 5 2		A ⊕ M → A	• • • • • R •
	EORB		C8 2 2	D8 3 2	F8 4 3	E8 5 2		B ⊕ M → B	• • • • • R •
Shift left arithmetic	ASLA	48 2 1						A ← 0	• • • • • 6 •
	ASLB	58 2 1						B ← 0	• • • • • 6 •
	ASL				78 6 3	68 7 2		M ← 0	• • • • • 6 •
Shift right arithmetic	ASRA	47 2 1						A ← C	• • • • • 6 •
	ASRB	57 2 1						B ← C	• • • • • 6 •
	ASR				77 6 3	67 7 2		M ← C	• • • • • 6 •
Shift right logical	LSRA	44 2 1						A ← 0	• • • • • 6 •
	LSRB	54 2 1						B ← 0	• • • • • 6 •
	LSR				74 6 3	64 7 2		M ← 0	• • • • • 6 •
Rotate left	ROLA	49 2 1						A ← C	• • • • • 6 •
	ROLB	59 2 1						B ← C	• • • • • 6 •
	ROL				79 6 3	69 7 2		M ← C	• • • • • 6 •
Rotate right	RORA	46 2 1						A ← C	• • • • • 6 •
	RORB	56 2 1						B ← C	• • • • • 6 •
	ROR				76 6 3	66 7 2		M ← C	• • • • • 6 •
Compare accumulators	CBA	11 2 1						A - B	• • • • • • •
Compare	CMPA		81 2 2	91 3 2	B1 4 3	A1 5 2		A - M	• • • • • • •
	CMPB		C1 2 2	D1 3 2	F1 4 3	E1 5 2		B - M	• • • • • • •
Compare index register	CPX		8C 3 3	9C 4 2	BC 5 3	AC 6 2		X _H - M, X _L - (M+1)	• • • • • • •
Test (zero or minus)	TSTA	4D 2 1						A - 00	• • • • • R R
	TSTB	5D 2 1						B - 00	• • • • • R R
	TST				7D 6 3	6D 7 2		M - 00	• • • • • R R
Bit test	BITA		85 2 2	95 3 2	B5 4 3	A5 5 2		A ← M	• • • • • R •
	BITB		C5 2 2	D5 3 2	F5 4 3	E5 5 2		B ← M	• • • • • R •
Branch	BRA						20 4 2	TEST	• • • • • • •
Branch if carry clear	BCC						24 4 2	C = 0	• • • • • • •
Branch if carry set	BCS						25 4 2	C = 1	• • • • • • •
Branch if overflow clear	BVC						28 4 2	V = 0	• • • • • • •
Branch if overflow set	BVS						29 4 2	V = 1	• • • • • • •
Branch if equal to zero	BEQ						27 4 2	Z = 1	• • • • • • •
Branch if greater or equal to zero	BGE						2C 4 2	N ⊕ V = 0	• • • • • • •
Branch if greater than zero	BGT						2E 4 2	Z + (N ⊕ V) = 0	• • • • • • •
Branch if less than zero	BLT						2D 4 2	N ⊕ V = 1	• • • • • • •
Branch if less than or equal to zero	BLE						2F 4 2	Z + (N ⊕ V) = 1	• • • • • • •
Branch if not equal to zero	BNE						26 4 2	Z = 0	• • • • • • •
Branch if minus	DMI						2B 4 2	N = 1	• • • • • • •
Branch if plus	BPL						2A 4 2	N = 0	• • • • • • •
Branch if higher	BHI						22 4 2	C + Z = 0	• • • • • • •
Branch if lower or same	BLS						23 4 2	C + Z = 1	• • • • • • •

OP = Operation Code

MC = Number of MPU Cycles

PB = Number of Program Bytes

Instruction	Mnemonic	Addressing Modes						Boolean/Arith Operation	Condition Reg				
		Implied	Direct	Immediate	Extended	Indexed	Relative		S	O	D	I	0
		OP MC PB	OP MC PB	OP MC PB	OP MC PB	OP MC PB	OP MC PB		H	I	N	Z	V
Branch to subroutine	BSR						8D 8 2	See Special Operations	•	•	•	•	•
Jump to subroutine	JSR				BD 9 3	AD 8 2			•	•	•	•	•
Jump	JMP				7E 3 3	6E 4 2			•	•	•	•	•
Return from subroutine	RTS	39 5 1							•	•	•	•	•
Return from interrupt	RTI	3B 10 1							•	•	•	•	•
Software interrupt	SWI	3F 12 1						PC + 1 → PC	•	•	•	•	•
Wait for interrupt	WAI	3E 9 1							•	•	•	•	•
No operation	NOP	02 2 1							•	•	•	•	•
Clear	CLRA	4F 2 1							•	•	R	S	R
	CLRB	5F 2 1							•	•	R	S	P
	CLR				7F 6 3	6F 7 2		00 → M	•	•	R	S	P
Clear carry	CLC	0C 2 1						0 → C	•	•	•	•	R
Clear interrupt mask	CLI	0E 2 1						0 → I	•	•	R	•	•
Clear overflow	CLV	0A 2 1						0 → V	•	•	•	R	•
Set carry	SEC	0D 2 1						1 → C	•	•	•	•	S
Set interrupt mask	SEI	0F 2 1						1 → I	•	•	S	•	•
Set overflow	SEV	0B 2 1						1 → V	•	•	•	S	•

CONDITION CODE SYMBOLS:

- H Half-carry from bit 3;
- I Interrupt mask
- N Negative (sign bit)
- Z Zero (byte)
- V Overflow, 2's complement
- C Carry from bit 7
- R Reset Always
- S Set Always
- † Test and set if true, cleared otherwise
- Not Affected

LEGEND:

- OP Operation Code (Hexadecimal);
- MC Number of MPU Cycles;
- PB Number of Program Bytes;
- + Arithmetic Plus;
- Arithmetic Minus;
- Boolean AND;
- MSP Contents of memory location pointed to by Stack Pointer;
- + Boolean Inclusive OR;
- ⊕ Boolean Exclusive OR;
- M Complement of M;
- Transfer Into;
- 0 Bit = Zero;
- 00 Byte = Zero;

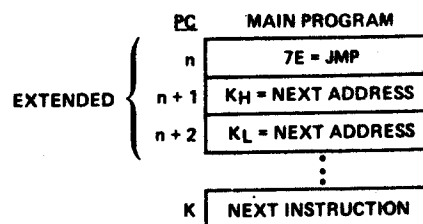
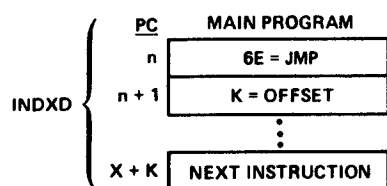
Note – Accumulator addressing mode instructions are included in the IMPLIED addressing

CONDITION CODE REGISTER NOTES:

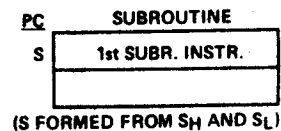
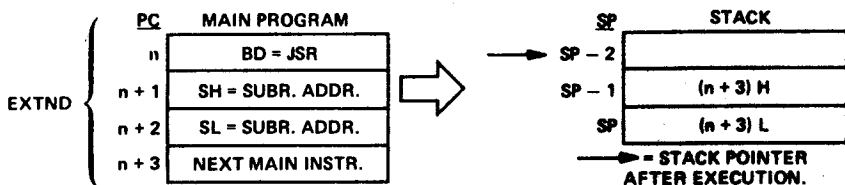
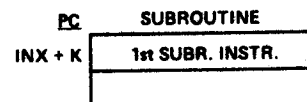
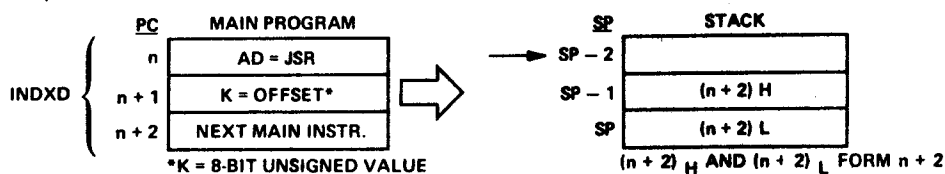
- (Bit set if test is true and cleared otherwise)
- 1 (Bit V) Test: Result = 10000000?
- 2 (Bit C) Test: Result = 00000000?
- 3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
- 4 (Bit V) Test: Operand = 10000000 prior to execution?
- 5 (Bit V) Test: Operand = 01111111 prior to execution?
- 6 (Bit V) Test: Set equal to result of N • C after shift has occurred.
- 7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
- 8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
- 9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
- 10 (ALL) Load Condition Code Register from Stack. (See Special Operations)
- 11 (Bit I) Set when interrupt occurs, if previously set, a Non-Maskable Interrupt is required to exit the wait state.
- 12 (ALL) Set according to the contents of Accumulator A

SPECIAL OPERATIONS

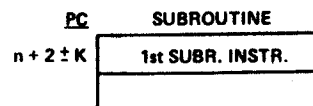
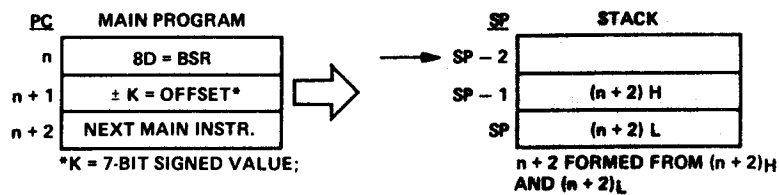
JMP, JUMP:



JSR, JUMP TO SUBROUTINE:

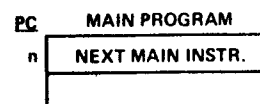
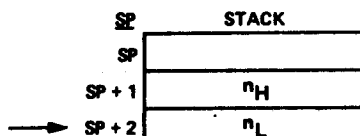
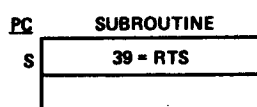


BSR, BRANCH TO SUBROUTINE:

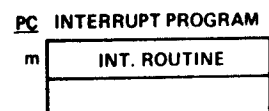
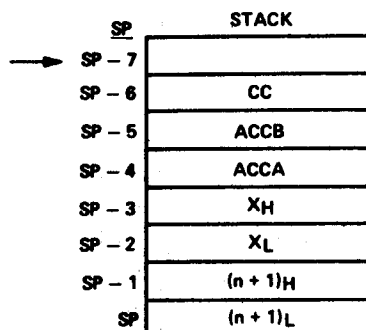
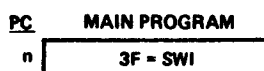


SPECIAL OPERATIONS

RTS, RETURN FROM SUBROUTINE:

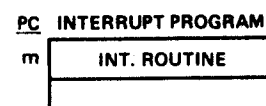
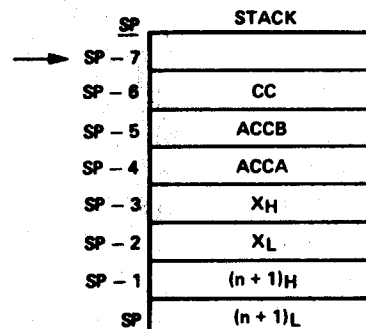


SWI, SOFTWARE INTERRUPT



m_H = (H - 0005)
m_L = (H - 0004)
H = ADDRESS WITH
ALL ADDRESS LINES
IN HIGH STATE

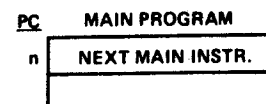
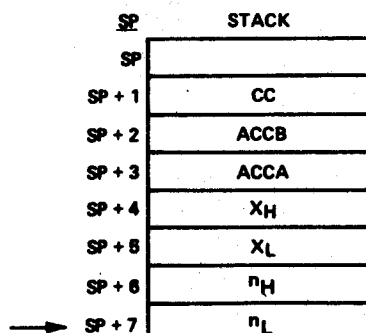
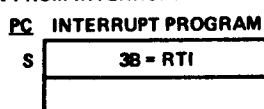
WAI, WAIT FOR INTERRUPT

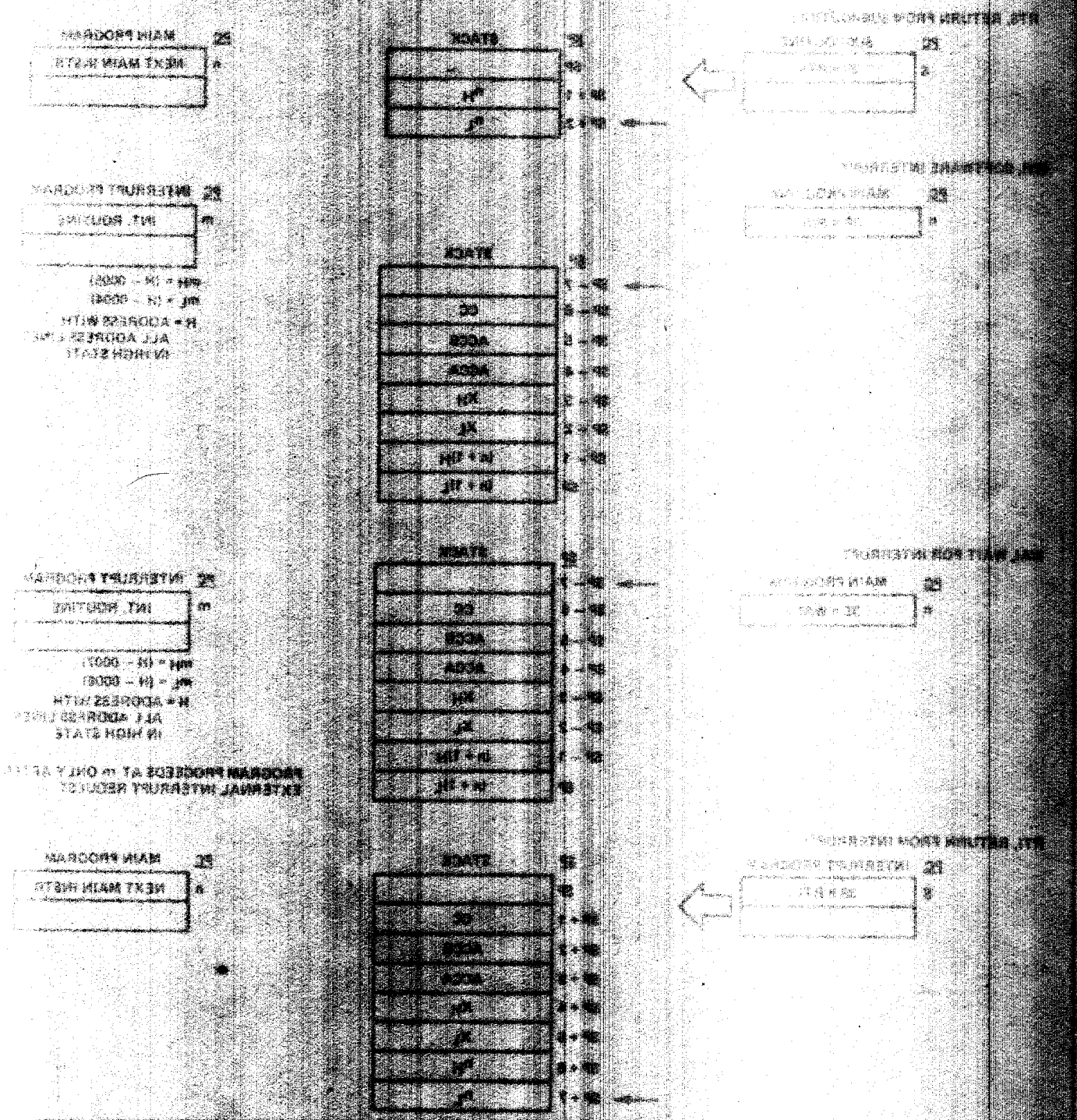


m_H = (H - 0007)
m_L = (H - 0006)
H = ADDRESS WITH
ALL ADDRESS LINES
IN HIGH STATE

PROGRAM PROCEEDS AT m ONLY AFTER
EXTERNAL INTERRUPT REQUEST

RTI, RETURN FROM INTERRUPT:





AMI Sales Offices

DOMESTIC

Western Area

1104 Highland Avenue
Suite I
Manhattan Beach, California 90266
Tel: (213) 379-2452

3031 Tisch Way
Suite # 202
San Jose, CA 95128
Tel: (408) 249-4550

Central Area

500 Higgins Road
Elk Grove Village, Illinois 60007
Tel: (312) 437-6496

725 S. Central Expressway
Suite C-5
Richardson, Texas 75080
Tel: (214) 231-5721

29200 Vassar Avenue
Suite # 214
Livonia, Michigan 48152
Tel: (313) 478-9339

Fox Meadows Office Building
3030 Harbor Lane North
Suite # 101
Minneapolis, Minnesota 55441
Tel: (612) 559-9004

24200 Chagrin Blvd.
Suite # 352
Cleveland, Ohio 44122
Tel: (216) 292-6850

Eastern Area

20 Robert Pitt Drive
Room # 212
Monsey, New York 10952
Tel: (914) 352-5333

1420 Providence Turnpike
Room # 220A
Norwood, Massachusetts 02062
Tel: (617) 762-0726

Altamonte Centre
249 No. Maitland Avenue
Suite # 317
Altamonte Springs, Florida 32701
Tel: (305) 830-8889

Axewood East
Butler & Skippack Pikes
Suite # 3A
Ambler, Penn. 19002
Tel: (215) 643-0217

INTERNATIONAL

England

AMI Microsystems Ltd.
108A Commercial Road
Swindon, Wiltshire
Tel: Swindon 31345

France

AMI Microsystems S.A.R.L.
124, Avenue de Paris
F-94300 Vincennes
Tel: 374 00 90

Italy

AMI Microsystems S.p.A.
via Pascoli 60
I-20133 Milano
Tel: 29 37 45

West Germany

AMI Microsystems GmbH
D-8 Muenchen 80
Rosenheimer Strasse 30
Suite # 237
Tel: 48 30 81

Japan

AMI Japan Ltd.
Daiwa Bank Building
1-6-21, Nishi Shimbashi
Minato-ku, Tokyo 105
Tel: (501) 2241

ERRATA

as of June 1977

PROTOTYPING BOARD MANUAL

SUPPLEMENT -- June 1977

The attached pages correspond to Revision D of the EVK Series. In particular, the schematics and board layout guides in the manual are obsolete, and should be replaced with the Revision D information.

READ ONLY MEMORY

The EVK is now supplied with one 2K-byte ROM containing PROTO. By convention, this ROM goes into Position 10, although Position 11 has the same partially decoded address. This leaves Position 11 open for some other ROM, either 2K (AMI S6831) or 1K (AMI S6830).

A separately-priced ROM appropriate for this usage is AMI's Micro Assembler/Disassembler. The MA/D program is a "direct" or "zero-pass" assembler which translates instruction mnemonics and hex values into RAM contents, and it can also reverse the process. The order number for the MA/D ROM is C10224.

For owners of old EVKs with PROTO on two ROMs, the new one-ROM PROTO is also available as a separately-priced item. The part number is C11003. This ROM is addressed at F000, while MA/D is address at E800.

PROTO COMMANDS

Although it is not stated in the manual, the Addr parameter for the G command may be omitted, in which case the value of P currently stacked is used.

It should be noted that any G command unstacks P (and the other registers) from the user's own stack, based upon the value of S which PROTO has saved in locations FFF2-FFF3. This pair of locations is not initialized by PROTO at power-up time. Therefore, a G command will not function properly after power-up until one of three events has taken place: at least one use of the Reset button (which will initialize the user's own stack at FF8F, shared with PROTO itself); or the use of the PROTO command S to force a value into FFF2-FFF3; or the direct entry (via the Reset Vector Switches) into a user program which initializes the user's own stack pointer.

The description in the manual of the E command is incorrect. What this command generates is an "S9" type hex format record, as described on Pages A-1 and A-2 of the manual. This is the End-of-Tape record expected by the PROTO command L and by similar 6800 software.

RS CUBED ROUTINES

The description of interface conventions for the RS CUBED routines in Chapter 4 may not be clear to programmers working at machine code level (or with MA/D), as opposed to users of AMI's macro assemblers.

The technique for entering any RS CUBED routine is to set up the 6800 registers as specified, then execute this sequence:

```
SWI
BYTE  hh
      xxx  xxx
```

In this sequence, "hh" represents a hex value as shown in the Index column of the RS CUBED write-ups, and "xxx" represents the beginning of the user's own executable code to which control returns after the RS CUBED routine has accomplished its function.

Using MA/D, the sequence would be entered this way, at, for example, RAM location 100:

```
.....:@100
0100:SWI
0101:hh
0102:xxx xxx
```

It should be noted that the assembly listings on Pages C-3 through C-48 reflect the contents of the PROTO ROM from F000 through F7FF. In other words, the LOC column on these pages should be offset by F000 to match the actual ROM.

OPERATING PROCEDURES

The edge connector supplied with the EVK is an Amphenol 261-10043-2, although the more expensive 225-805-43 mentioned in Chapter 5 may also be used.

The general logic of the EVK draws about 3.5 amps, not the 4.0 amps specified in Chapter 5.

The protocol used by the TTY interface is 7 bits, with 2 stop bits, even parity. This may be seen from the way that PROTO sets up the ACIA.

The current loop set-up in the manual is incorrect. A +5V level should be tied to DCD (B63) and CTS (B65). The rest of the wiring is:

<u>TTY</u>	<u>EVK</u>	<u>SIGNAL</u>
7	B64	OUT +
6	B66	OUT -
4	B68	IN +
3	B70	IN -

The switch directions shown on Page 20 of the manual are for toggle switches, and are therefore backward for slide switches like those on the EVK 300. Use the board markings when you install jumpers or slide switches.

Whenever the EVK goes through Reset, either because of power-up or because the Reset button has been used, remember that the Reset Vector Switches must point to appropriate code, such as the start of PROTO, at F000 (binary 1111 0000 0000 0000).

Part Description	EVK-98	EVK-99	EVK-100	EVK-200/300
P.C.B. P/N 03-0013-000 Rev D	1	1	1	1
S6800	0	1	1	1
S6810-1	~	4	4	8
S6820		1	0	3
S6834		0	0	1/4
S6850		1	1	1
9263-001 C11003		1	1	1
555		0	1	1
1488		~	0	1
1489A			1	1
4702			1	1
74S00			2	2
74S02			2	2
7404			1	1
7407			1	1
74S08			1	1
74S10			2	2
74S20			1	1
74LS30			2	2
74S30			1	1
74S32			1	2
7437			0	1
7438			1	1
74S138			3	3
74160			0	3
74S257			2	2
8T97B			14	14
96S02			1	2
Socket 8 Pin C-9308-02			1	1
Socket 14 Pin C-9314-02	0	0	16	19
Socket 16 Pin C-9316-02	~	~	21	25

Part Description	EVK-98	EVK-99	EVK-100	EVK-200/300
Socket 24 Pin C-9324-02	↑	↑	6	15
Socket 24 Pin (R.N.) TS-61024			0	1
Socket 40 Pin C-9340-02			1	4
Capacitor .1 uf, 16 V + UK 16-104			30	37
Capacitor .01 uf, 6 V +			0	1
Capacitor, Tantalum, 22 uf, 15 V			5	7
Capacitor, Tantalum, 4.7uf, 75 V,			0	1
Capacitor, Mica, 8 pf, 500 V			4	4
Capacitor, Mica, 56 pf, 500 V			2	2
Capacitor, Mica, 100 pf, 500 V			0	2
Capacitor, Mica, 560 pf, 500 V			0	1
Capacitor, Mica, 270 pf, 500 V			2	2
Capacitor, Mica, 470 pf, 500 V			0	2
Transistor, MJE 340			0	1
Transistor, 2N3563			1	1
Transistor, 2N4402			1	1
Transistor, 2N5400			0	1
Transistor, 2N5771			2	2
Transistor, 2N5772			2	2
Diode A15F			1	3
Diode 1N914			3	3
Diode 1N4003			6	7
Resistor 1M 1/4 W			2	2
Resistor 1K 1/4 W			6	9
Resistor 1.5 K 1/4 W			3	4
Resistor 2K 1/4 W			2	2
Resistor 3.3K 1/4 W			0	4
Resistor 4.7K 1/4 W			0	16
Resistor 10K 1/4 W	0	0	0	1
Resistor 10 1/4 W	↑	↑	0	1
Resistor 22 1/4 W			1	1

Part Description	EVK-98	EVK-99	EVK-100	EVK-200/300
Resistor 22K $\frac{1}{4}$ W	↑	↑	0	1
Resistor 33 $\frac{1}{4}$ W			1	1
Resistor 51 $\frac{1}{4}$ W			2	2
Resistor 100K $\frac{1}{4}$ W			3	3
Resistor 100 $\frac{1}{4}$ W			1	1
Resistor 120 $\frac{1}{4}$ W			1	1
Resistor 150 $\frac{1}{4}$ W			1	1
Resistor 150K $\frac{1}{4}$ W			0	1
Resistor 470 $\frac{1}{4}$ W			2	4
Resistor 510 $\frac{1}{4}$ W			1	1
Resistor Packs 8 Res. 16 Pin 4.7K Beckman 898-3-R4.7			2	2
Amphenol Connector (86 Pin) 261-10043-2			2	2
8 Position Switch CTS-206-008			2	2
4 Position Switch CTS-206-004			0	1
Switch 1101C			0	4
Switch 8125A			1	1
Crystal 1.000 MHZ			0	1
Crystal 2.4576 MHZ			1	1
Potentiometers, 15 Turn, 20K, Burns 3006P-1-203			2	2
Cap Cer Disc .47 μ fd, > 6 volts			1	1